OVERCOMING TCP DEGRADATION IN THE PRESENCE OF MULTIPLE INTERMITTENT LINK FAILURES UTILIZING INTERMEDIATE BUFFERING

THESIS

Duane F. Harmon, Major, USAF

AFIT/GE/ENG/07-11

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/07-11

OVERCOMING TCP DEGRADATION IN THE PRESENCE OF MULTIPLE INTERMITTENT LINK FAILURES UTILIZING INTERMEDIATE BUFFERING

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

Duane F. Harmon, BSEE

Major, USAF

March 2007

AFIT/GE/ENG/07-11

OVERCOMING TCP DEGRADATION IN THE PRESENCE OF MULTIPLE INTERMITTENT LINK FAILURES UTILIZING INTERMEDIATE BUFFERING

Duane F. Harmon, BSEE

Major, USAF

Approved:

Scott R. Graham	27 Feb 07
Maj Scott R. Graham, PhD (Chairman)	Date
Kenneth M. Hopkinson	27 Feb 07
Dr. Kenneth M. Hopkinson (Member)	Date
[illegible]	27 Feb 07
Capt Richard R. Beckman, PhD (Member)	Date

Acknowledgments

I would like to thank my family for placing their lives on hold for the 21 months required to complete the AFIT academic program. Their continual love and support grounded me during the entirety of this academic assignment.

I would also like to thank the AFIT faculty, especially Maj Scott Graham and Dr. Kenneth Hopkinson for their insight, re-vectoring, and assistance during the completion of this thesis. Maybe an old dog can learn new tricks.

Duane F. Harmon

Table of Contents

List of Figures

Page

List of Tables

AFIT/GE/ENG/07-11

Abstract

It is well documented that assumptions made in the popular Transmission Control Protocol's (TCP) development, while essential in the highly reliable wired environment, are incompatible with today's wireless network realities in what we refer to as a challenged environment. Challenged environments severely degrade the capability of TCP to establish and maintain a communication connection with reasonable throughput. This thesis proposes and implements an intermediate buffering scheme, implemented at the transport layer, which serves as a TCP helper protocol for use in network routing equipment to overcome short and bursty, but regular, link failures. Moreover, the implementation requires no modifications to existing TCP implementations at communicating nodes and integrates well with existing routing equipment. In a simulated six-hop network with five modified routers supporting four challenged links, each with only 60% availability, TCP connections are reliably established and maintained, despite the poor link availability, whereas 94% fail using standard routing equipment, i.e., without the TCP helper protocol.

OVERCOMING TCP DEGRADATION IN THE PRESENCE OF INTERMITTENT LINK FAILURES UTILIZING INTERMEDIATE BUFFERING

I. Introduction

1.1 Background

Information superiority is achieved only when timely and accurate information is placed in the hands of the warfighter who needs it. Such distribution of information can only be realized by networking every soldier, sailor, and airman into a vast network, spanning the globe. The Department of Defense (DoD) vision for a global network is realized in the concept of the Global Information Grid (GIG). The GIG is the globally interconnected end-to-end (ETE) set of information systems, processes, and personnel for collecting, storing, processing and disseminating information to it's personnel and automated systems [1]. Information sharing and near real-time information has become a force multiplier and as outlined in the 2006 DoD Chief Information Officer Strategic Plan [2], the DoD is transforming to become a Net-Centric force. This is a departure from the traditional platform and organization centric operations of the past. The impetus for the transition is the evolutionary increase in available information and the ever increasing need of the warfighter to access near real-time data for situational awareness and mission accomplishment. The ultimate goal of the transition to a Net-Centric force is ensuring timely and accurate information is available to the correct person (or machine), in any place, at the proper time.

The hardware infrastructure required to support the Net-Centric force is necessarily a hybrid of wired and wireless domains. Wired infrastructure at stateside and forward encampments must communicate with deployed forces using wireless communication devices. Supporting the hybrid network are hundreds of individual communication protocols coexisting in harmony to provide a robust and reliable network over which information is collected, requested, and disseminated. In the modern military, mobility is a key requirement and forward deployment to harsh environments with no infrastructure is common. At these forward tactical edge locations, the reliability and performance of the network is stressed via the use of notoriously unreliable wireless communications. To become truly Net-Centric, the underlying network architecture must adapt to and overcome the physical realities of the unreliable wireless medium and ultimately provide a reliable communications infrastructure to the warfighter with minimal restrictions.

1.2 Problem Statement

Communication requires the successful transmission of data between two points, or nodes, within the network. At the tactical edge of the GIG, where the warfighter is deployed, communication is frequently required with a node out of immediate range and thus reachable only through forwarding the message through several "hops." Data must traverse many point-to-point links to reach the intended destination. Several of these point-to-point links will be wireless, making traditional ETE communication problematic, especially if several of the links are challenged. As shown in Figure 1.1, when multiple challenged links exist in an ETE communication path, the probability of an uninterrupted path drops significantly with the increase in number of challenged links present. For example, if four wireless hops are required and each hop is experiencing only 90% availability, then the probability of an ETE uninterrupted path is only 65.6%.

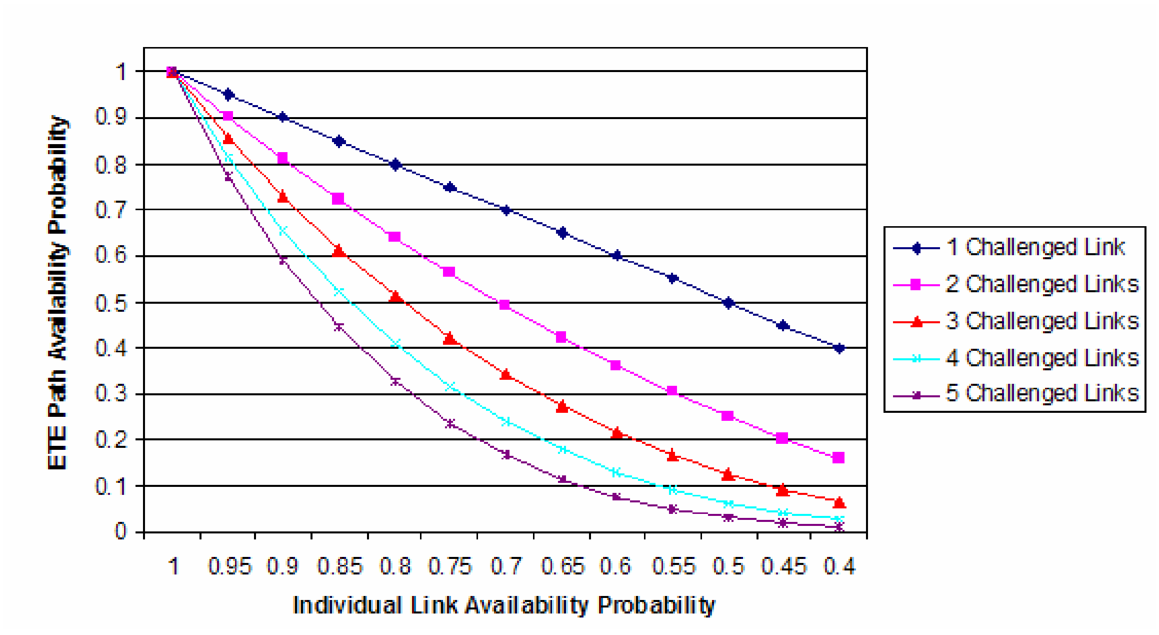


Figure 1.1: ETE Path Probability for Multiple Challenged Links

The dominant method of communication over the GIG will be via packet switched networks using the ubiquitous Internet Protocol (IP), and the well known Transmission Control Protocol (TCP), an ETE protocol primarily designed to ensure reliable delivery of packets. TCP has been highly optimized based on assumptions specific to wired networks. Key among these assumptions is the fact that loss in wired networks is primarily a result of congestion at routers, as opposed to bit errors. In wired links, bit error rates are often measured in magnitudes of 10^{-8} or 10^{-9}. In stark contrast, wireless communication bit errors rates approach 10^{-2} (or even 10^{-1} in some cases) as a result of spectrum interference, fading channels and other naturally occurring sources of noise. Unfortunately, packet errors due to such events are erroneously interpreted as network congestion by TCP, causing a reduction in transmission attempts precisely when they may be needed most. As a result, data throughput in the wireless domain is significantly degraded. Compounding this reality is the fact that current network architecture does not provide any intermediate buffering of packets which have successfully traversed earlier links and are currently experiencing difficulty overcoming a particular link. Thus, the probability of successful ETE transmission of packets in the unreliable wireless medium drops off precipitously.

This research focuses on the use of intelligent intermediate buffers to overcome individual point-to-point wireless link transmission errors, effectively hiding localized non-congestion errors from the TCP connection endpoints and preventing the reduction of data throughput for the ETE TCP communication. Additionally, the desirable feature of not requiring modifications to the communicating TCP endpoints is maintained to avoid requiring a specific TCP implementation.

1.3 Research Approach

A relatively new area of study is that of so-called challenged networks. In a challenged network, connections between nodes are frequently disconnected and reconnected, possibly due to environmental conditions or even on a scheduled basis. An example of the latter is a non-stationary satellite that has known windows of availability. In challenged networks, the throughput of TCP connections is severely degraded due to periods of link non-availability, referred to here as *link-wink* [3]. In a military environment, *link-wink* could be the result of many environmental factors such as jamming, spectrum interference, aircraft turbulence, or covertness.

Wireless network throughput can be improved by correcting the erroneous reduction in TCP transmission attempts using three general strategies; ETE proposals, split-connection proposals, and link layer proposals [4]. ETE proposals modify the network layer TCP protocols to explicitly notify the TCP sender of congestion before invoking TCP *congestion control*. Split-connection proposals mask *congestion control* invoking triggers from the TCP sender by performing some form of buffering and filtering action at intermediate nodes. Link layer proposals improve link layer protocols for reduced error rates and local resend actions.

This research combines aspects of split-connection and link-layer schemes to challenged networks with short periods of non-availability. These schemes are extended to accommodate ETE connections with multiple wireless hops, any of which could be severely challenged with low availability or high bit error rates. Specifically, TCP-aware transport layer buffering using a split-connection scheme over each challenged wireless link is evaluated. In effect, each router connecting a wireless link along the ETE TCP connection acts as a local proxy for the TCP communication endpoints.

1.4 Assumptions, Limitations, and Resulting Implications

Many forms of traffic will coexist over the GIG. This investigation targets File Transfer Protocol (FTP)-like TCP traffic and large file transfers between two communication points. Other forms of communication such as User Datagram Protocol (UDP) traffic or bi-directional communication utilizing TCP are not specifically investigated, but are also expected to benefit from intermediate buffering. The communication channel investigated in this thesis assumes disruption periods can be very frequent, but only for short durations on the order of milliseconds to seconds. Longer duration *link-wink* is non conductive to TCP connections and requires fundamental modifications to the TCP timeouts which drive the timescale over which TCP can effectively function. However, the applied approach applies to transfers experiencing longer duration outages if TCP's time constants are extended. Jain et al. [5], present a thorough discussion of the routing and connection support implications for very-long duration (hours or days) disconnected nodes and situations where an ETE path may never exist.

A key feature of this research is that an attempt is made to overcome TCP's limitations in an environment where TCP is extremely likely to fail. TCP is used by a majority of applications and any expectation of a readily accepted modification to TCP for battlefield use is unlikely. Accordingly, the proposed scheme within this thesis requires no modifications to the TCP protocol or the communicating endpoints as it focuses on adding complexity to the network routing infrastructure rather than the edge. This research suggests that modifications to the endpoints will improve performance further, but such modifications are left for future research.

It is assumed that a route between the communicating endpoints exists, at least on an intermittent basis. No communication scheme can overcome a lack of communication path. Static routing is utilized in this research; however the developed model should be applicable to other routing schemes, perhaps with some adaptation. It is also assumed that the data and acknowledgment communication paths are symmetrical, which is reasonable for short duration transfers. The nodes in this research are stationary, yet no mobility limitations are placed on the model. The developed model can be applied to mobile scenarios if properly matched routing and link-layer protocols are utilized. Of final note is the assumption that *link-wink* does not invoke routing re-establishment algorithms since the outage duration is for a short period only and will be available again momentarily.

1.5 Summary

This chapter outlines the motivation and limitations of this thesis research. Chapter Two provides a review of pertinent concepts, further details of the problems of TCP in a challenged environment, and a review of literature applicable to this research. Chapter Three presents a detailed discussion of the developed model along with the design decisions implemented within the model. Chapter Four describes the methodology used in this research, experiments utilized, analysis of results, and conclusions concerning buffering of TCP data streams in a challenged network. Chapter Five summarizes the motivation, research methodology, results and observations, conclusion of this research, and offers suggestions for future research activities.

II. Literature Review

2.1 Chapter Overview

This chapter provides the reader with a brief introduction to the background knowledge required for a thorough understanding of this research. It is assumed that the reader has a general knowledge of computers and computing networking. In-depth discussion of the specific domain of this research is contained herein as well as specific parameters of interest. A brief overview of the network protocol stack is presented, followed by a detailed discussion of the network layer TCP protocol. The concept of a challenged network is then introduced with a discussion of TCP shortcomings in such an environment. A discussion of some published research pertaining to this thesis is then introduced and followed by a discussion of this thesis research and its contributions.

2.2 Network Protocol Stack

A network, simply defined, is a collection of communicating entities connected together by communication links. The communicating entities or communication links need not be homogeneous, however each communicating pair must share a common communication protocol to communicate effectively. Modern networks may consist of millions of communicating devices and intermediate nodes, using various types of communication links and a vast array of communication protocols. In order to provide a structure for network design, development, and maintenance, network designers have defined a protocol stack comprised of various layers. This thesis focuses on the internet protocol stack shown in Figure 2.1. Each layer of the protocol stack interacts with the layer immediately above or below it. A higher layer uses the services of layers beneath it, and provides service to layers above it.

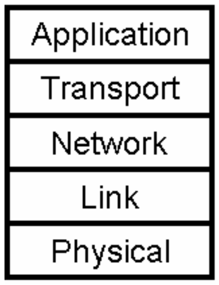


Figure 2.1: The Internet Protocol Stack

2.2.1 Application Layer

The application layer is the "raison de existence", or why the network exists at all. Typical examples of applications are web browsing, email, file sharing, or teleconferencing. These applications are located on hosts which communicate with one another, via the network, using protocols such as HTTP(web browsing), FTP(file transfer), or SMTP(e-mail). Communicating applications exchange information via messages using the transport layer. This research is presented using the client-server communication model; however it applies equally well to peer-to-peer and hybrid communication models.

2.2.2 Transport Layer

The transport layer exists to transport messages between the applications located on the hosts at the network endpoints. To an application, the transport layer abstracts away the communication details and behaves as if it were directly connected to the communication partner. Messages can be quite large and must be broken into segments for efficient delivery. For the message to be understood, every segment may need to be delivered. The internet uses TCP and UDP to transport messages either reliably or unreliably between hosts. The transport layer uses the services of the network layer to perform its obligations.

2.2.3 Network Layer

Hosts within a network can be separated by a vast distance with many intermediate nodes, called routers, between them. The network layer is charged with moving each transport layer segment from the source host to the destination host by forwarding (or routing) it in the most efficient manner possible. Each segment is encapsulated within a network layer entity called a datagram. Thus, the network layer is charged with finding a suitable path, composed of individual links, and moving datagrams from the source host to the destination host. Each intermediate node must determine where to forward a datagram by examining the destination address and choosing an outbound link which will move the datagram closer to its destination. By "closer", we often mean topologically closer, as opposed to physically closer.

The dominant network layer protocol is the well-known Internet Protocol (IP) and is essentially an addressing scheme where a host address is also a unique identifier of that host. The IP addressing scheme implies a hierarchical topology, where a static routing table suffices. To adapt to link failures, it is necessary to incorporate some redundant links, and to implement adaptive routing protocols that sense the presence or absence of links and respond by changing the routing tables. As links fail, successful routes may or may not be discovered by the network layer, leaving no guarantee of successful datagram delivery. Hence the requirement for the transport layer to provide its own reliable message delivery guarantee between hosts.

There are many routing protocols in existence. This research is not concerned with the routing protocols and merely assumes static routing. Hence, no particular routing protocol need be identified.

2.2.4 Link Layer

After the Network Layer determines which node to forward the packet to, the link layer provides a mechanism to move datagrams between adjacent nodes. The link layer uses the services of the physical layer, which provides an abstraction of a "bit pipe" in which bits are introduced at one end and received on the other end. It is the responsibility of the link layer to handle bit errors, presenting the abstraction of an error free packet delivery system from one node to another.

Frequently, multiple nodes share a common link; e.g., a radio channel, or a multi-tapped bus. This is especially true at the extremities of a network. Such an arrangement offers connection flexibility, but introduces the possibility of packet collisions, and must therefore have a mechanism, or protocol, to allocate the use of the channel. For increased performance, nodes can also be connected via a dedicated link, which avoids the possibility of collisions. A datagram may encounter many types of individual links on its path from source to destination. The specific protocol, such as Ethernet, ATM, or PPP, is chosen to optimize the transmission medium and link properties. The model developed in this thesis uses PPP between adjacent nodes; however it is equally applicable to other link layer protocols.

2.2.5 Physical Layer

The physical layer is responsible for physically transmitting information between geographically separated nodes. Here, bits are transformed into some form of electromagnetic energy, which can be propagated from one node to another through a channel. Examples of channels include optical fiber, twisted-pair copper wire, free-space optical wireless, or radio frequency transmissions. Each of these requires differing protocols in order to function. However, each presents the abstraction of a “bit pipe” to the link layer above. The physical layer protocol is dependent on the type of link and transmission medium.

2.3 Transmission Control Protocol (TCP)

The transport layer can provide reliable or unreliable delivery of segments over a network layer, which does not provide any guarantee of datagram delivery. TCP, originally defined in RFC 793, is the connection-oriented transport layer protocol that guarantees reliable, in-order delivery of segments between communicating source and destination hosts, despite the underlying unreliable network services [6, 7]. An additional goal of TCP is to maintain some sort of general fairness between information flows.

TCP establishes an ETE connection between the source and destination hosts (communicating partners). During connection establishment, each host records state information pertaining to its communicating partner and maintains this information throughout the active connection. Specific connection state information consists of receive and send buffer size and content, sequence numbers and acknowledgement numbers, and *congestion control* parameters (timers and variables). Intermediate nodes of the network layer maintain no state information pertaining to a TCP connection.

A TCP connection is established via a three-way handshake between a client host, hereafter referred to as the receiver, and the server host, hereafter referred to as the sender. During the handshake, each host records the IP address and port number of its communication partner, establishes initial sequence numbers, and creates send and receive buffers. The sender's send buffer contains the data to be transferred to the receiver. The receiver's receive buffer is initially empty and is populated as segments are received. The receive buffer is emptied as in-order data is delivered to the application layer above.

The sender sends segments to the receiver and receives acknowledgments from the receiver. Acknowledgements are generally sent for every segment and are cumulative. The number of unacknowledged segments in transit is dynamic and is based on the sender's perception of the network congestion state as discussed below.

In a non-challenged network, segments generally arrive in order at the receiver and cumulative acknowledgements arrive in order at the sender. If segments are lost, subsequent segments may arrive at the receiver, creating a "gap" in the received sequence numbers. When a gap is detected in the sequence numbers, the receiver does not acknowledge the segments received after the gap, but rather continues to send duplicate cumulative acknowledgements for the next expected in-sequence byte number, i.e., the first segment lost in the gap. The sender identifies lost segments through receipt of duplicate acknowledgements. In addition, if all segments are lost, the sender must have some way of knowing this. Discovery of such an event is enabled via a timeout timer and the lack of any acknowledgements within a derived timeout interval triggers a series of retransmissions.

In the underlying unreliable network, segments and acknowledgments can be lost due to many factors such as spectrum interference, high bit error rates, packet collisions, or buffer overflows (due to congestion). Optimal TCP performance depends on its ability to estimate network performance and adapt. As TCP sends data and receives acknowledgments, it maintains estimates of round trip time (RTT) and RTT variance, which it then uses as a basis for various timers and timeouts. If TCP estimates the condition of the network poorly, it becomes sub optimal in one of two ways. Either it retransmits too often, congesting the network with duplicate messages, or it fails to transmit when conditions are favorable, losing the opportunity and delaying successful delivery of the message.

2.3.1 Sliding Windows

TCP incorporates several constructs in its quest to optimize network resources. The Sliding Window, referred to as *send window* is designed to prevent a sender from overwhelming either the receiver, or any intermediate link, by only allowing a limited number of packets outstanding, or “in flight.” The TCP sender maintains a dynamically adjusted *send window* that slides "to the right" as time elapses. The left edge of the window slides right as data is acknowledged by the receiver. The right edge of the window advances (to the right) as the receiver advertises available receive buffer space within an acknowledgement. Thus the total window width is set by the receiver, to prevent receiver buffer overflows, and is called the *offered window*. The receiver's receive buffer will be reduced by the occurrence of lost segments or mis-ordered segment arrivals. From the sender's viewpoint, the *offered window* contains sent, but not yet acknowledged segments, and the number of segments that can be sent immediately if allowed by the *congestion window* (*cwnd* as discussed in 2.3.2). TCP attempts to reach an optimal point at which the number of unacknowledged segments is equal to the *offered window*. In challenged networks, the fact that the amount of unacknowledged segments cannot exceed the *offered window* is a critical limiting factor in TCP performance.

2.3.2 Congestion Control

In addition to the aforementioned TCP features that ensure the receiver's buffer is not overwhelmed, TCP maintains a *congestion window* parameter, *cwnd*, to reduce congestion in the network and reduce the possibility of router buffer overflows enroute to the receiver. In effect, *cwnd* is a measure of the number of unacknowledged bytes that can be in transmission without causing network congestion. TCP utilizes *slow start*, *congestion avoidance*, *fast retransmit*, and *fast recovery* algorithms as part of the *congestion control* mechanism. Each of these algorithms modifies *cwnd* in differing ways. Figure 2.2 provides a graphical representation of the effect of *congestion control* algorithms on the *cwnd* parameter in response to congestion events.

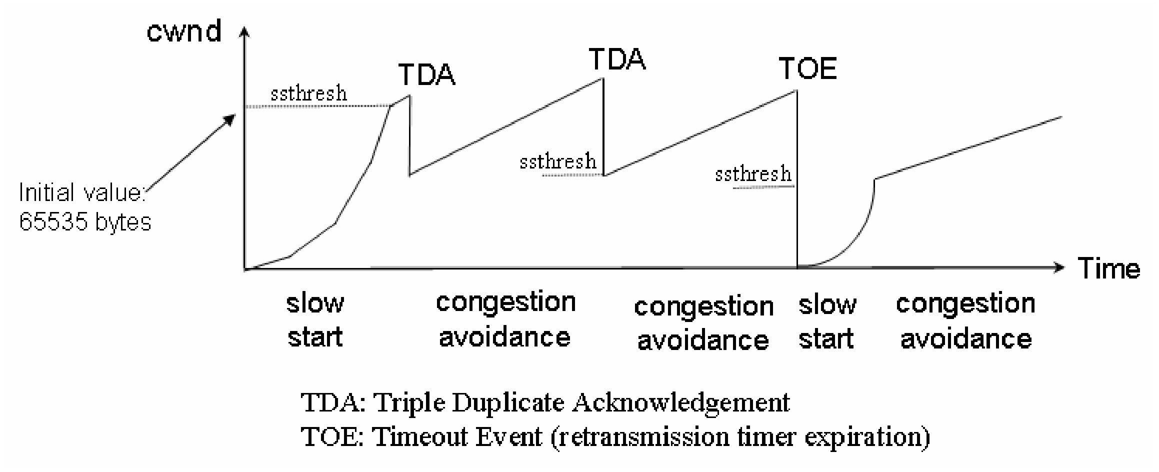


Figure 2.2: Congestion Control Algorithm Effects on *cwnd* Parameter

2.3.2.1 *<u>Slow Start and Congestion Avoidance</u>*

In classic TCP implementations, *cwnd* is initially set at one or two packets (1460 or 2920 bytes) and is incremented by 1460 bytes every time an acknowledgement is received. The upper bound of the *slow start* algorithm is reached when *cwnd* reaches the *slow start* threshold parameter, *ssthresh*, typically set at 65535 KB by default. The *slow start* ramp-up is often referred to as an exponential increase phase. Once the *cwnd* parameter has incremented to that of *ssthresh*, the TCP sender enters the *congestion avoidance* phase, where *cwnd* is incremented by 1460 bytes (the largest allowable datagram payload for Ethernet) each round trip time.

2.3.2.2 *<u>Fast Retransmission / Rapid Recovery</u>*

If three duplicate acknowledgments are received at the sender, the sender assumes a segment is lost and immediately tries to resend what is interpreted as a lost segment without waiting for the retransmission timer to expire. The *ssthresh* and *cwnd* parameters are set to one half of the *cwnd* value and the sender enters *congestion avoidance* immediately upon retransmission of the lost segment. *Slow start* is not invoked due to the fact that receipt of the duplicate acknowledgments indicates data is still arriving at the receiver.

2.3.2.3 Timeout and Retransmission

As a sender sends data, it sets a timeout time by which an acknowledgement should be received. The timeout period is set by using a current estimate of RTT with an allowance for variance. If an acknowledgment for outstanding segments has not been received by the timeout expiration, it is interpreted as a network trouble indication and the *cwnd* parameter is set to 1460 bytes while all unacknowledged segments are resent according to the *slow start* algorithm. Repeat timeouts result in an exponential backoff between retransmission attempts via a doubling of the timeout timer to a maximum of 64 seconds. If repeat timeouts persist for a period of 9 minutes, the TCP connection is terminated by the sender.

2.4 Challenged Environment

As mentioned previously, in the underlying unreliable network, TCP segments and acknowledgments can be lost due to many factors such as spectrum interference, high bit error rates, packet collisions, or buffer overflows (congestion). Regardless of the loss mechanism, segments are lost/dropped and the state information at the communicating endpoints changes. Ideally, the endpoints adapt properly to handle loss events. This is especially critical for the endpoint that is actively sending data and is expecting acknowledgements. This thesis investigates overcoming a challenged environment where each wireless link in a multi-hop wireless environment experiences high losses.

Thus, availability of the link is reduced and the opportunity for successful TCP communication is significantly degraded.

2.5 TCP Performance in a Challenged Environment

Well tuned to a high reliability environment, TCP performance degrades significantly in the presence of high link losses. TCP responds to lost data packets by invoking the *congestion control* mechanisms discussed previously. The response is based on assumptions that loss is the result of buffer overflow at intermediate routers between the source and destination of the TCP communication flow. TCP's *congestion control* mechanism incorporates a fairness doctrine and immediately reduces the load of intermediate routers by reducing its transmission attempts. In theory, other TCP connections using the same intermediate routers will implement their own *congestion control* mechanisms, rectifying the temporary overflow situation, and resulting in network wide recovery in a more or less fair fashion.

In a challenged environment, losses are more likely to be attributable to wireless communication difficulties and thus invocation of *congestion control* mechanisms is an improper response of the TCP sender. TCP is unable to discriminate between wireless communication difficulties and true network congestion, leading to the aforementioned degradation. Therefore, some implementation changes to TCP are required to operate effectively in the challenged environment.

Additionally, TCP requires a three-way handshake between an initiator and receiver for connection establishment. When links are non-challenged, connection establishment segments are easily received. In a heavy loss environment however, connection initiation segments can be easily lost, potentially resulting in a connection establishment delay, or even worse, a connection establishment abort.

2.6 Relevant Research

2.6.1 Selective Acknowledgement

The TCP Selective Acknowledgement (SACK) option, defined and documented in RFC 2018 [8], is designed to overcome multiple losses in a single transmission window by explicitly notifying the sender which segments in the byte stream have been delivered and which segments remain outstanding. In the SACK scheme, an acknowledgement with three contiguous blocks of received segments can be conveyed to the sender, allowing the sender to interpret which segments to resend. Ultimately, this reduces the recovery time in the event of multiple losses and reduces overhead of unnecessary retransmissions.

The SACK concept is utilized in this thesis to identify a single contiguous missing series of bytes, but only between specialized intermediate routers. No attempt is currently made to utilize SACK between the communicating endpoints.

As discussed in the sliding windows context, the amount of unacknowledged data in flight is upper bounded by the advertised *receive window*. The SACK concept allows a potential avenue to overcome this limit when intermediate proxy routers are used, but the TCP sender may require modification. Consideration was given to the possibility of allowing the specialized intermediate routers to send SACKs to the TCP sender, especially since the SACK option allows reneging, but concern over the impact to the TCP timer state data necessitated postponing this option. It is strongly encouraged that any follow-on work attempt to utilize SACK to its maximum potential, despite the requirement of TCP sender modification.

2.6.2 Snoop

The Snoop protocol [9] is a local loss enhancement in wireless networks that places a snoop agent in the wireless access point. Hence, the snoop agent sees all segments related to a communicating pair from connection establishment through connection termination. The Snoop protocol maintains per-connection state and caches packets locally for possible retransmission in the event of a loss and retransmits lost segments locally as required. Under this scheme, duplicate acknowledgements are handled locally if the missing segment is cached and duplicate acknowledgements are destroyed to keep the sender from misinterpreting congestion due to link layer loss. The key desirable feature of the Snoop protocol is that the ETE TCP semantics are not modified, however it does require that the SACK option be set for optimal performance. The authors' [9] simulations achieved speedups up to 20 times over "regular TCP."

For the interested reader, an 802.11 (WiFi) implementation of the Snoop protocol using the OPNET simulation suite was presented by Chi Ho Ng et al. [10]. The outlined implementation was for a single wireless link between the endpoint and wireless access point, as Snoop was intended to be used. The authors' [10] claim a significant TCP performance increase, up to 68 times in a 30% packet error rate environment, utilizing the Snoop protocol. These results are somewhat misleading as the implemented loss mechanism prevented key control segments loss.

Several features of the Snoop protocol are included within this thesis, however the Snoop protocol was designed for use in a single wireless access point. This thesis uses a modified Snoop protocol at each specialized intermediate router within the network, allowing for localized error recovery within the network itself, not simply in the final wireless hop. Additionally, this research makes no assumption that critical control segments such as connection establishment and termination segments are immune from loss.

2.6.3 Split TCP

Split TCP [11] was introduced for mobile ad hoc networks and TCP connections that suffer from mobility losses. In such an environment, route failures are common and the channel capture effect unfairly hinders rapidly changing TCP flows. Under Split TCP, each node acts as a proxy, accepting temporary custody of a segment via a local acknowledgement, and forwards the segment on to the destination. In effect, each ETE connection is split into several shorter localized TCP connections. Split TCP requires modifications to the TCP sender that allow more data to be in flight via an ETE window and a *congestion window*. The specific methods and algorithms utilized in this scheme were not discussed. The authors' [11] simulations show that fairness among TCP connections is increased and throughput of individual TCP connections increased by as much as 40%.

The forwarding concept of Split TCP that continually moves data toward the destination and the local acknowledgement are similar to this thesis, but the splitting of the ETE connections into multiple smaller connections and modifying the TCP sender differs greatly. Without implementation details, it is difficult to compare and contrast the approaches further.

2.6.4 TCP Bulk Repeat

TCP Bulk Repeat [12] is an ETE scheme proposed for improving TCP performance in a heavy loss environment. In this scheme, the TCP sender is modified by performing a bulk retransmission of all outstanding packets in the *send window* in the event of a loss, setting a fixed retransmission timeout rather than the standard exponential backoff used in TCP-Reno, and restricting a reduction of the *cwnd* parameter to error-induced cases only. Error induced loss is discriminated from congestion by using an expected data rate calculation and noting the difference from the achieved rate. Under the TCP Bulk Repeat scheme, throughput is increased by shortening the recovery period after a loss event, but at the cost of a higher overhead of unnecessarily resent segments. The authors' [12] simulations show an increase in throughput performance of TCP Reno and TCP Westwood by an order of magnitude in high error rate (>5%) cases with the most notable performance increase in bursty error cases.

TCP Bulk Repeat differs greatly from this research primarily due to the fact that the scheme modifies the TCP sender only, where necessary TCP connection state information is available for calculations and modification. This thesis research however applies changes to intermediate nodes in the network.

The TCP Bulk Repeat concept of retransmitting all non-acknowledged packets in the *send window* is used in this thesis, but only in the case of a timeout event. This thesis could benefit by applying the fixed retransmission timeout scheme rather than using an exponential backoff approach, but such a modification is left for future consideration due to fairness concerns to other existing TCP flows.

2.6.5 Strategic Buffering

Reynolds introduces mathematical models of TCP FTP transmission time over a single challenged link with and without strategic buffering at intermediate routers [3]. The models, as well as testbed simulations, showed that strategic buffering within the network could reduce TCP transmission time by handling loss events closer to the source of loss. The implementation used in the simulations used an inefficient bulk repeat retransmission scheme that resends all buffer contents every 10 msec. The retransmission scheme assumes dedicated use of the challenged link and is inherently unfair in a shared medium environment. Additionally, the scheme modified the TCP sender to interpret intermediate acknowledgments from strategic buffering routers similar to the selective acknowledgment scheme discussed earlier.

This study is a continuation effort derived from the initial work of Reynolds. The problem formulation and concept of adding complexity to the network routing mechanisms to accommodate strategic buffering is similar. This work however, investigates multiple challenged links in a simulation environment with refined retransmission mechanisms that do not assume dedicated use of a link, i.e., this approach is capable of fairness, although the extent of fairness achieved has not been studied. Additionally, this effort makes no modifications to the TCP sender, adding only complexity to the network that is transparent to TCP implementations.

2.7 Summary

This chapter provides a short introduction to TCP performance and how it degrades in a challenged environment. Several published works that address TCP's performance were discussed and their similarity to and differences from this work were highlighted. The primary contribution of this work is to develop and evaluate a model for supporting FTP-like TCP communication in a multiple hop scenario with multiple challenged links. This is the first known attempt to improve and analyze TCP performance in a multiple hop environment where each link has significantly reduced availability, perhaps as low as 60 percent.

III. Model Description

3.1 Chapter Overview

This chapter introduces the relevant design details of the model built for this thesis. It should be noted that the reader can skip this chapter if the design philosophy and specific implementation details are not of interest.

The model was developed using OPNET 12.0 and includes many of the details outlined in Snoop TCP with concepts from SACK, Split TCP, and TCP Bulk Repeat. The driving scenario for this model is a TCP connection over several challenged wireless hops between communicating endpoints where traditional TCP implementations will simply fail. Much of the model is focused on reliability between successive hops and adherence to a custodial buffering principle, without any modification to the TCP sender and receiver. Increasing reliability between successive hops requires introducing complexity to the network while preserving the ETE semantics of existing TCP implementations.

3.2 OPNET Modeler

OPNET Modeler [13] is a simulation tool for modeling and simulation of computer networks. OPNET modeler provides a discrete event simulation engine, a graphical user interface, and hundreds of basic device models that can be utilized and modified as needed for network simulation as well as providing the ability to create custom models for research activities.

3.3 Model Introduction

This study analyzes the value of adding reliability at the network layer via a transport layer “helper” protocol. A key performance question is whether the increase in TCP byte stream throughput and decrease in ETE delay in a challenged link environment (when compared to standard network implementations) justifies the additional cost and complexity of such a network.

The developed protocol is considered a transport layer helper protocol. The protocol is transport layer aware as it requires access to network layer datagram header information and acts on TCP segments contained within them. Additionally, the developed protocol supports the transport layer TCP protocol using the services provided by the network layer. The support reinforces transport layer reliability and provides a mechanism to detect and recover from loss events. However, the support does not violate the ETE requirements of TCP and the TCP endpoints are unaware of the presence of the helper protocol.

The developed model supports networks such as the one shown in Figure 3.1. This network is representative of a modern-day military network environment where forward deployed ground and air forces, remote sensors platforms, and loitering aircraft communicate in a shared medium wide area network. Reach-back to higher levels of command is also provided by wireless links and the network contains some wired infrastructure.

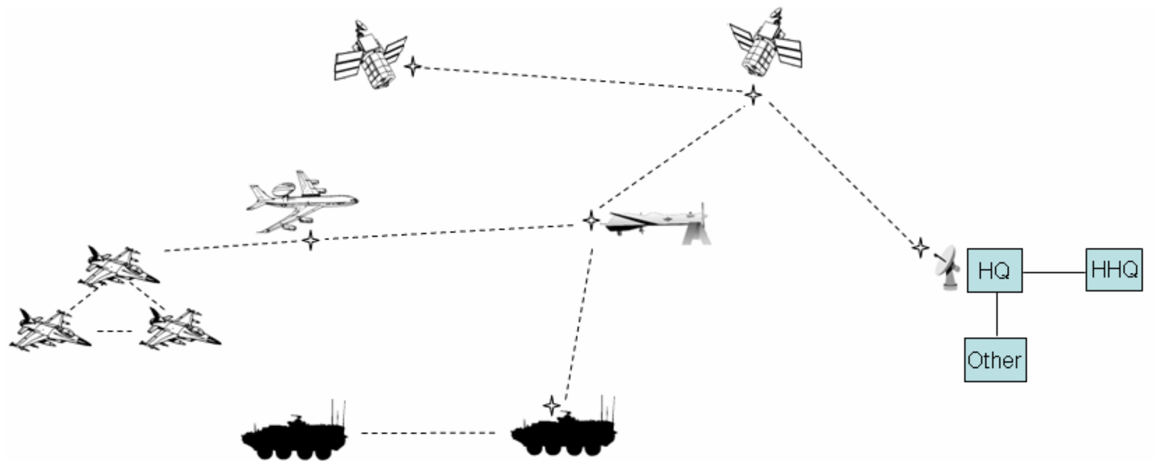


Figure 3.1: Example Challenged Network Topology

The complexity within the network is introduced via customized routers, symbolized by a star pattern in the figure. It is envisioned that such a router is introduced at the end of wireless backbone links, enabling localized recovery due to challenged link characteristics. Thus, for the topology displayed in Figure 3.1, a specialized router would be located in satellites, AWACS and UAV platforms, the satellite control station link, and strategic tracked ground vehicles.

3.4 Model Requirements

Specific model implementation is discussed in detail throughout this chapter, however high level model requirements are highlighted here to provide a context for the remaining detailed model discussion.

As discussed in Chapter Two, non-congestion losses (channel failures) in a wireless network are perceived by a TCP sender as congestion losses. When losses are improperly categorized as congestion, TCP invokes its *congestion control* algorithms, reducing throughput and extending the transmission period of TCP communication. The goal of the developed model is to insulate an unmodified TCP sender from non-congestion losses while still preserving ETE TCP semantics. Non-congestion loss events should be handled as close to the source of loss as possible without intervention from the TCP sender. This is achieved through the use of specialized routers, referred to here as **intermediate link proxies** due to their ability to act on behalf of a sender. The use of these proxies along a multi-hop intermittent path should increase the reliability of the ETE connection and enable TCP communication in a degraded environment where TCP would otherwise fail due to timeout conditions, exponential retransmission backoff, and eventual unconditional connection termination.

Intermediate link proxies will require high-speed memory be available within the host router for buffering TCP segments. Potentially hundreds of simultaneous TCP flows could be utilizing a **link proxy**, hence memory usage by a particular TCP flow should be minimized such that unnecessary segments are removed from a local cache as quickly as possible. Accordingly, some means of communication should exist between **link proxies** that provides intermediate acknowledgements (accepting custody of the datagram) for immediate feedback and subsequent release of upstream memory. This is best accomplished via custom intermediate acknowledgement packets between **link proxy** routers.

Detection of loss events should occur as quickly as possible to facilitate rapid recovery, increase link utilization rates, and improve individual TCP flow throughput. A **link proxy's** immediate proximity to the potentially challenged link can be used to monitor the link itself via estimates of round trip time to the next **link proxy** router in the transmission flow direction as well as an estimate of round trip time to the transmission endpoint. Variance of the round trip time can be determined and used to minimize the impact of slight variations in round trip time and the negative impact of premature timeout handling.

For optimal performance after a loss event, a **link proxy** should respond immediately by resending the lost data (from its local cache) to the destination on behalf of the source. In a shared medium, the recovery of a loss event should be "polite" in some sense, taking care not to dominate the use of the link and degrade multiple TCP flows passing over the link. Politeness is introduced via an exponential backoff mechanism between repeated resend events, implemented in the same manner in which a TCP sender politely backs off due to a congestion event. This mechanism allows buffered segments from each TCP flow over the link to be resent, providing maximum fairness to each flow trying to utilize the degraded link.

Duplicate acknowledgements that indicate a loss event should be destroyed (when loss events are locally detected and locally handled) to eliminate the possibility that the TCP sender will receive them and perceive the loss as congestion, thereby reducing its *congestion window*. The net effect of detecting the loss locally, handling the required lost segment retransmission, and destroying the duplicate acknowledgments is complete masking of the non-congestion channel failure from the TCP sender. The sole exception to this case is SYN-ACK segments that are used during TCP connection establishment.

Such segments do not invoke *congestion control* algorithms and should always be forwarded when observed.

In a challenged environment, it is highly probable that acknowledgements from the TCP receiver are lost. A **link proxy** should not cache such acknowledgement segments, as doing so would imply the requirement to resend them in the event of no intermediate feedback between **link proxy** routers. Blind retransmission of TCP receiver acknowledgments can potentially invoke TCP sender *congestion control* and its resulting performance drop. A better mechanism for handling lost TCP receiver acknowledgments is for the **link proxy** router to store a small amount of acknowledgment state information for the flow and determine if acknowledgements are lost via comparing any newly received receiver acknowledgment information with stored state data. When lost acknowledgments are detected, a **link proxy** should regenerate them, stimulating the TCP sender into increasing the *congestion window* and allowing for increased sender throughput.

It should be highlighted that under no circumstances should **link proxy** routers generate TCP receiver segment acknowledgments without first discovering that the receiver has in fact acknowledged the segment in question. Doing so would violate the ETE semantics of TCP, erroneously advancing the TCP sender *send window*.

3.5 Proxy Router Architecture

The architecture of typical low-cost routers is a central processor with small queues on the inbound and outbound links as shown in Figure 3.2. All IP datagrams arriving at the router are inspected and routed by a single processor. Once the outbound link is determined, the switching fabric routes the datagram to the proper outbound link for transmission. Datagrams can be dropped within the router by queues reaching their size limitations due to an arrival rate of datagrams in excess of the service rate of the central routing processor or outbound link transmissions.

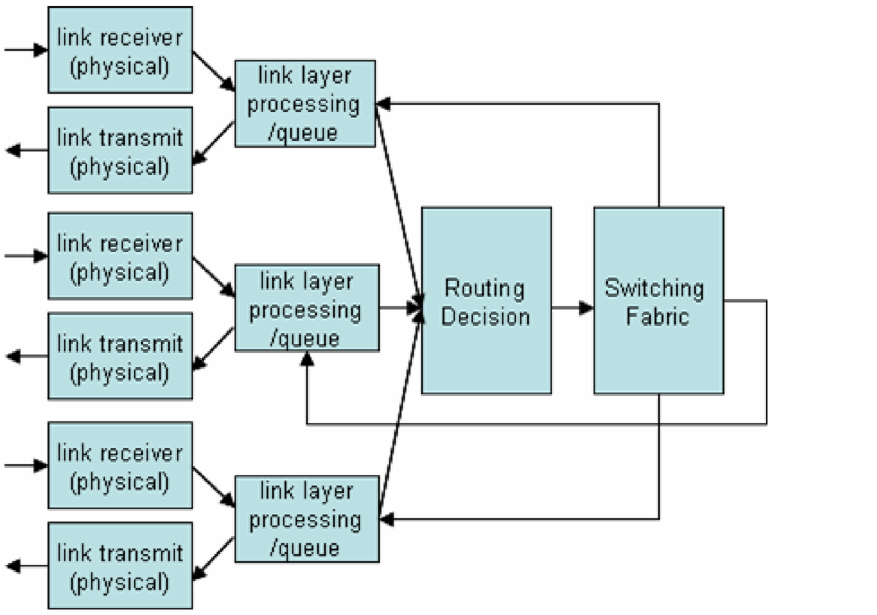


Figure 3.2: Central Processor Router Architecture

Two central routing processor router architectures were investigated for use this research. The first architecture is shown in Figure 3.3 and features a single proxy processor that works in concert with the central routing processor. This architecture features a single simple proxy approach that works in tandem with the network layer routing. Using this architecture allows easier design and software coding since only a single module is required in the router that provides all TCP flow proxy capability and memory management. The centralized proxy architecture could become a severe performance bottleneck however, especially in the presence of hundreds of high throughput TCP flows.

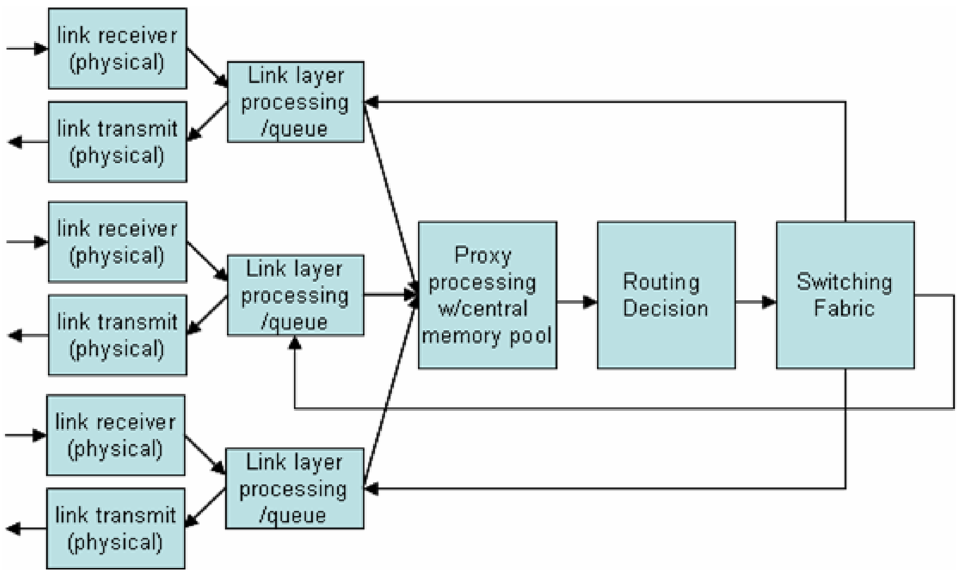


Figure 3.3: Central Processor Router Architecture with Central Proxy

The distributed proxy architecture displayed in Figure 3.4, places a proxy on each link, hence the reference to proxies as "**link proxies**". The architecture increases the model complexity and increases the amount of state data that must be maintained for TCP flow support. The distributed architecture was used in this study for two primary reasons. First, a future inclusion of a distributed memory pool management algorithm that will best utilize limited memory resources within the router can be easily evaluated using the developed model. Second, additional research at AFIT is investigating optimal inter-router buffer management algorithms. The distributed **link proxy** scheme can interface with such an algorithm for follow-on research.

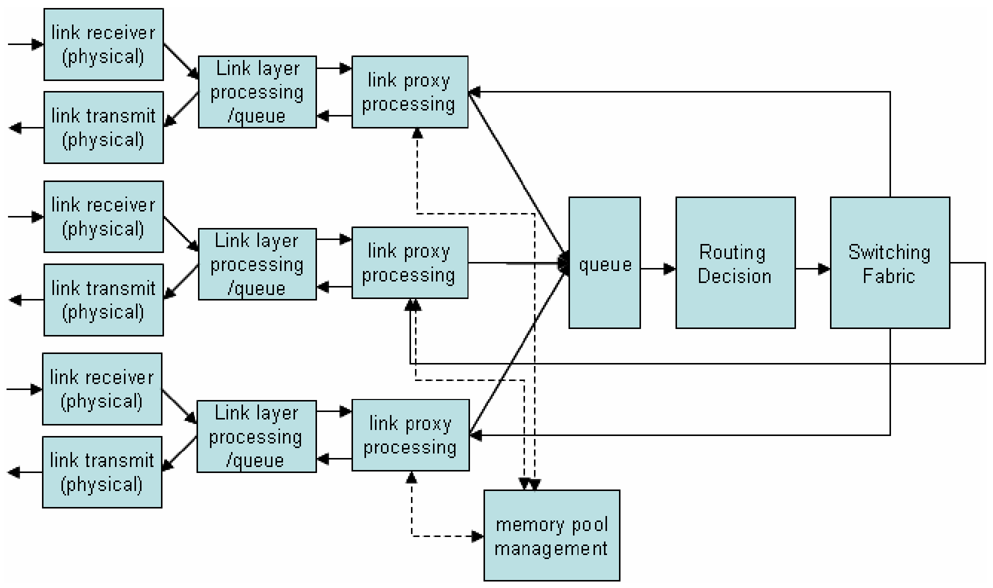


Figure 3.4: Central Processor Router Architecture with Link Proxies

The OPNET model for the customized **link proxy** router is shown in Figure 3.5. The function of the router is unchanged and the existing standard network layer IP routing remains unchanged from the standard OPNET model. Within the standard router, four of the link layer PPP links have been modified by the insertion of a transport layer **link proxy** between the link receiver/transmitter pair and the IP routing process. In essence, every datagram designated to travel over the link is inspected by the **link proxy.** A decision is made to process the datagram, if any action is necessary, or simply forward the datagram immediately. A centralized memory pool manager is included to provide efficient memory pool management with limited centralized memory resources. The memory pool manager communicates only with the **link proxies** within the router. The transmitter/receiver pairs model the physical layer and are unchanged from the standard OPNET PPP model.

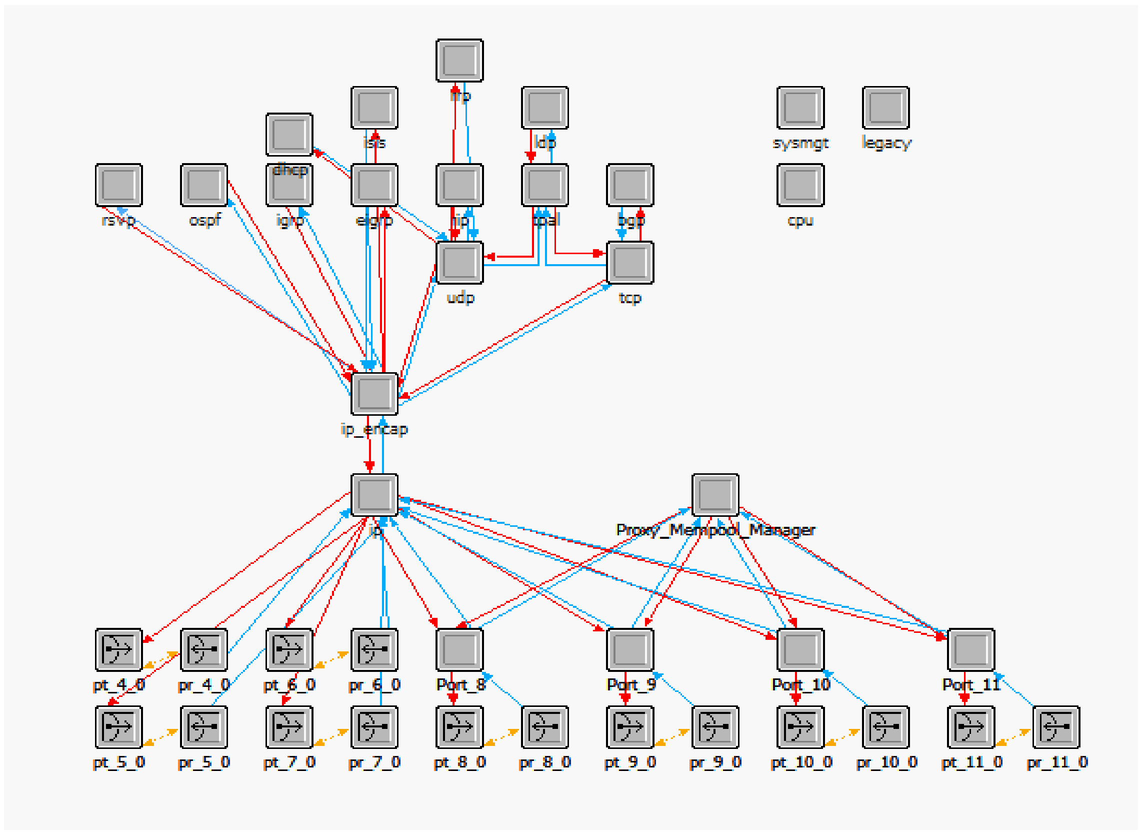


Figure 3.5: Custom OPNET Router

3.6 Intercepted Packet Formats

The location of the **link proxy** allows every datagram passing over the PPP link to be intercepted and potentially acted upon by the **link proxy**. This thesis research focuses strictly upon FTP-like TCP connections and thus simply forwards all other traffic unhindered. The model architecture allows for action upon other communication protocols as well, but such expansion is left for follow-on research.

3.6.1 IP Datagram Packet Format

Every datagram coming from or destined for the PPP link carries an IP header as shown in Figure 3.6. A **link proxy** requires some critical information from the IP header in order to determine proper processing actions. Critical IP header field data includes the protocol identifier (identifying the type of payload carried by the datagram), 32 bit source and destination IP addresses of the datagram originator and intended receiver, and the datagram length.

Protocol identifiers 0-137 and 253-255 are currently utilized (TCP is identified by a value of six), leaving 138-252 for potential use. The developed model assumes that a new protocol identifier can be assigned to allow proxy capable routers to communicate with one another. The particular assignment number used in the developed model is 150 and is defined in OPNET external header file ip_higher_layer_proto_reg_sup.h for global recognition.

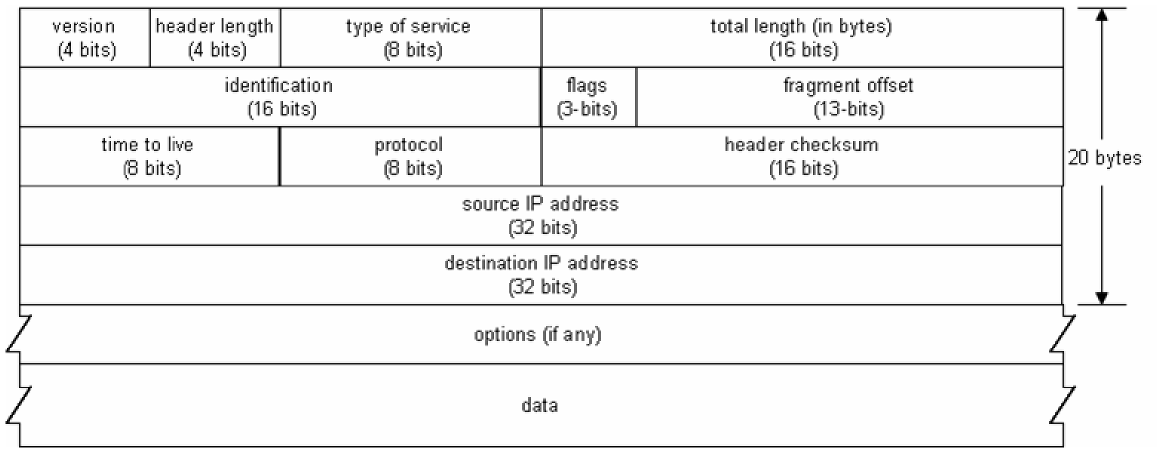


Figure 3.6: IP Header Format

3.6.2 TCP Segment Packet Format

The standard TCP header format is shown in Figure 3.7. TCP connections are ETE, so a TCP connection is uniquely identified by the port number of the source and destination hosts found in the TCP header and the source and destination IP addresses found in the IP header. All information within the TCP header is considered critical for **link proxy** use and will be discussed in detail as it is used in the model. The encapsulated TCP segment data is not inspected by **link proxies** and is simply buffered with the TCP segment itself.

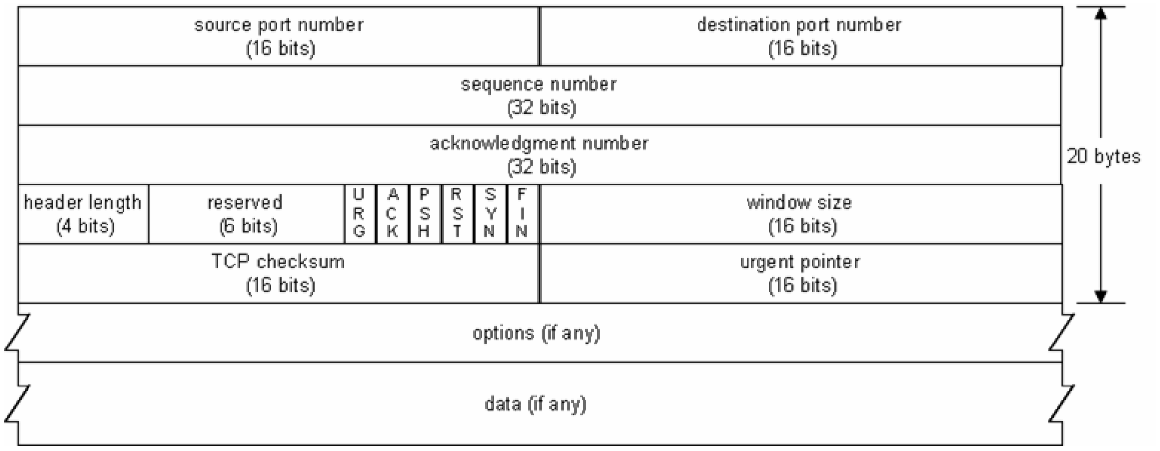


Figure 3.7: TCP Header Format

3.6.3 Proxy Packet Format

Communication between proxy capable routers is carried within IP datagrams with only the data carried in the header format identified in Figure 3.8. Three types of messages are conveyed between adjacent proxy routers. The most common message is an intermediate acknowledgment, or IACK, which is generated upon receipt of new data and travels back toward the data sender to be intercepted by the **link proxy** that last handled the TCP segment. Persist query messages are sent downstream toward the TCP receiver to query flow status and probe for lost acknowledgments. Persist response messages are generated upon receipt of a query and are sent upstream towards a TCP source to be intercepted by the query initiator. While a small amount of overhead is introduced as a result of the inter-proxy messages, the impact is minimal and ultimately improves inter-proxy performance for overcoming loss events.

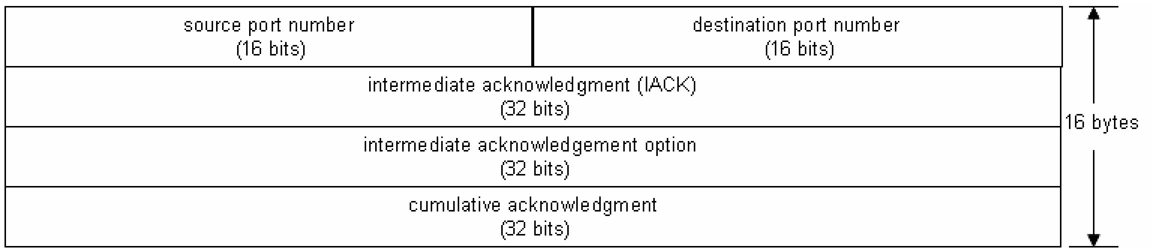

source port number
(16 bits)
destination port number
(16 bits)
intermediate acknowledgment (IACK)
(32 bits)
intermediate acknowledgement option
(32 bits)
cumulative acknowledgment
(32 bits)
16 bytes

Figure 3.8: Proxy Header Format

3.7 Link Proxy

The **link proxy** process model is displayed in Figure 3.9. The Init state instantiates the process with proper initial state variables, both user defined and standard, and creates data structures necessary for **link proxy** algorithm execution. Upon completion of the Init state, a **link proxy** enters the Wait state until an event requiring action occurs. Departure from the wait state is dependent on the discrete event that awakened the **link proxy** for action. Datagram arrival transitions the model to the Handle_Packet state where the **link proxy** determines what action, if any, should be performed on the arriving datagram. If the arriving datagram requires memory allocation from the memory pool, the **link proxy** enters the Await response state in anticipation of a memory allocation response message from the memory pool manager, follow-up handling of the initiating datagram, and a return to the wait state. Unsolicited message arrival directly from the memory pool manager transitions the model to the Manager_msg state where the **link proxy** takes appropriate action and returns to a wait state. The Handle_Timer state is entered upon the expiration of a self-scheduled interrupt event, returning to the wait state upon timer event action completion. Finally, the Endsim state is activated at the completion of a simulation for graceful memory recovery.

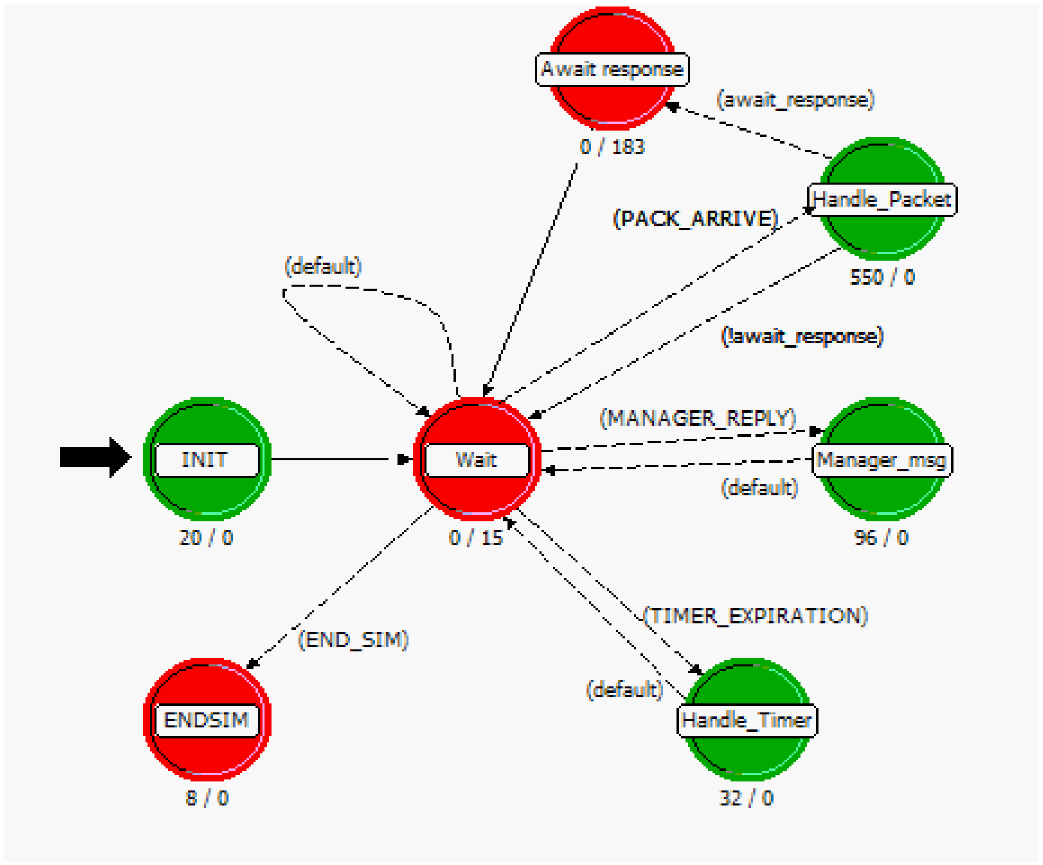


Figure 3.9: Link Proxy Process Model

3.7.1 Init State

A **link proxy** is initialized by a wake-up event initiated by the memory pool manager. The wakeup message contains a pointer to the particular **link proxies'** shared memory allocation structure built by the memory pool manager. Upon receipt of the wake-up, the Init state identifies itself and its parent process and reads its settable attributes for the simulation. The **link proxy** then initializes internal statistics variables and creates a list structure for storing TCP connection records.

3.7.2 Wait State

The wait state is a simple holding point for awaiting the arrival of a discrete event that requires attention in the **link proxy**. Interrupt types are classified as stream, self, or end of simulation interrupts. Any interrupt to the **link proxy** process model must fall into one of the three categories and the process model passes to the Handle_Packet (stream), Handle_Timer(self), or Endsim(endsim) states depending on interrupt type.

3.7.3 Handle_Packet State

Link proxies inspect every datagram that passes through them, however not every datagram requires proxy actions to be performed. Table 3.1 highlights the datagram payload types that require processing by the **link proxy** according to the input interface on which the datagram is received. The appropriate handling routine (handle_data or handle_ack) is also identified in the table. If the datagram carries a TCP or proxy message payload, then the Handle_Packet routine determines what flow the arriving datagram is attributed to and processes or forwards the segment according to the TCP flags, proxy message type, presence of data, and arriving interface. If the arriving segment is the first encountered for a particular flow, a flow record is constructed to maintain per-connection state information for all future segments passing through the **link proxy**. If the arriving datagram carries any other payload type other than TCP or proxy message, then the datagram is immediately forwarded with no further action.

Messages from the memory pool manager are not classified or handled within the Handle_Packet state, but are included in table 3.1 for completeness.

Table 3.1: Link Proxy Action Decision Matrix

Payload	**Type**	**Arriving Interface**	**Link Proxy Action**
TCP segment	SYN	From ip (routed)	Process (handle_data)
TCP segment	SYN	From link receiver	Forward
TCP segment	SYN-ACK	From ip (routed)	Process (handle_data)
TCP segment	SYN-ACK	From link receiver	Process (handle_ack)
TCP segment	DATA	From ip (routed)	Process (handle_data)
TCP segment	DATA	From link receiver	Process (handle_ack)
TCP segment	ACK	From ip (routed)	Forward
TCP segment	ACK	From link receiver	Process (handle_ack)
Proxy message	IACK	From ip (routed)	Forward
Proxy message	IACK	From link receiver	Process (handle_ack)
Proxy message	Persist Query	From ip (routed)	Process (generate persist response)
Proxy message	Persist Query	From link receiver	Forward
Proxy message	Persist Response	From ip (routed)	Forward
Proxy message	Persist Response	From link receiver	Process (handle_ack)
Memory Pool Manager message	All	From manager	Process
Other (not listed)	N/A	Any	Forward

3.7.4 Await_response State

The await response state is a by-product of using a distributed **link proxy** architecture model with a central memory pool scheme. If a datagram arrives carrying a TCP segment that should be cached, the **link proxy** cannot simply allocate new memory for cache purposes and must solicit a memory allocation from the memory pool manager. The solicitation and response is performed within the router, currently modeled with no delay, and the **link proxy** completes the handle_data routine from within the Await_response state.

3.7.5 Manager_msg State

It is anticipated that more TCP flows will pass through a proxy router than memory can be allocated for. Should the memory pool manager decide that the amount of memory allocated to a specific flow within the **link proxy** needs to be reduced to meet other demands, the method of conveying the reduction is a message arrival from the manager and entry into the Manager_msg state. Should this event occur, then TCP segments currently cached in excess of the reduced allocation will be purged according to a purging policy yet to be developed, and localized loss event recovery will not be possible for the purged segments. The specifics of how to handle purged packets is left for follow on research, but the mechanisms for doing so have already been included.

3.7.6 Handle_Timer State

Upon expiration of a timeout or persist timer, the model will transition to the Handle_Timer state where a determination must be made of which timer event expired. Additionally, many TCP flows can be passing through the same router, so a determination of the particular TCP flow for which the timer expiration event occurred must be determined. Once these two critical pieces of information are determined, the **link proxy** either generates a persist query message to probe the link, resends all cached TCP segments for the flow, or deletes a flow connection record, depending on timer type.

3.7.7 Connection Record

Link proxies can support multiple simultaneous TCP connections and state information is maintained for every communicating pair. The unique 4-tuple of source and destination IP addresses and port numbers is sufficient to discriminate among flows. Key information is maintained for a flow in a connection record that records the known state of a flow for loss event recovery and connection management purposes. Of note is that a **link proxy** must maintain the most recently noted sequence numbers and data lengths for data traveling from both source to destination and destination to source in order to generate valid acknowledgement messages on behalf of the destination when acknowledgments are lost. Additional recorded state information consists of the last in-sequence sequence number and data length, acknowledgement numbers (including IACKs), and *receive window*s. Timer events, derived from maintained estimates for RTT and RTT variance both to the next **link proxy** (if it exists) and the destination are also maintained for loss event discovery and recovery purposes.

3.7.8 Proxy_Handle_Data Routine

TCP's Connection establishment segments (SYN and SYN-ACK) and data segments are intercepted and cached by **link proxies** before forwarding them on the link.

As noted in Table 3.1, only routed datagrams destined for an outbound link are acted upon by the handle_data routine of a **link proxy**.

A flow chart of actions performed in the proxy_handle_data routine on datagrams meeting the Table 3.1 requirements is displayed in Figure 3.10. Every received segment for a TCP flow is compared to currently cached segments and only those segments not already acknowledged by the destination are forwarded. Should a **link proxy** intercept a segment already acknowledged, due to a sender timeout event, then the proxy destroys the datagram and generates an acknowledgement on behalf of the destination. Only one acknowledgement may be generated for a particular segment because creation of multiple identical acknowledgments will be interpreted as duplicate acknowledgments, invoking sender *congestion control* algorithms. If the received segment has already been acknowledged via an IACK from a downstream **link proxy**, but not yet acknowledged by the destination, then the **link proxy** forwards the segment without caching it.

If the received segment is a new segment and memory has been allocated by the central memory pool manager for buffering the TCP flow, then the segment is cached and forwarded. An IACK is then generated and sent upstream toward the last **link proxy** to cache the segment. Under normal circumstances, all received segments will be in-sequence, however it is possible that segments will be received out of order due to loss events. In these circumstances, the generated IACK will include a selective acknowledgement field that conveys the cumulative IACK and the first post-gap byte received.

If a received segment is already cached, indicating that an IACK or ACK has not been received upstream, then the cached segment state information is updated as having been received and forwarded multiple times for use by the handle_ack routine. The segment is then forwarded and an IACK is generated for the upstream **link proxy**.

To facilitate loss event detection, timeout and persist events are scheduled if a timeout event does not exist. Unlike the Snoop protocol, the timeout timer is not extended (i.e. incrementally increased) with every forwarded segment, it need only exist. Extending the timeout timer with every forwarded segment would mask loss events longer than desired in a severely challenged environment where links are down for even short durations.

Finally, if the intercepted segment header carries a FIN flag, indicating that the sender has nothing more to send, an internal state flag is set for use by the handle_ack routine for flow record termination.

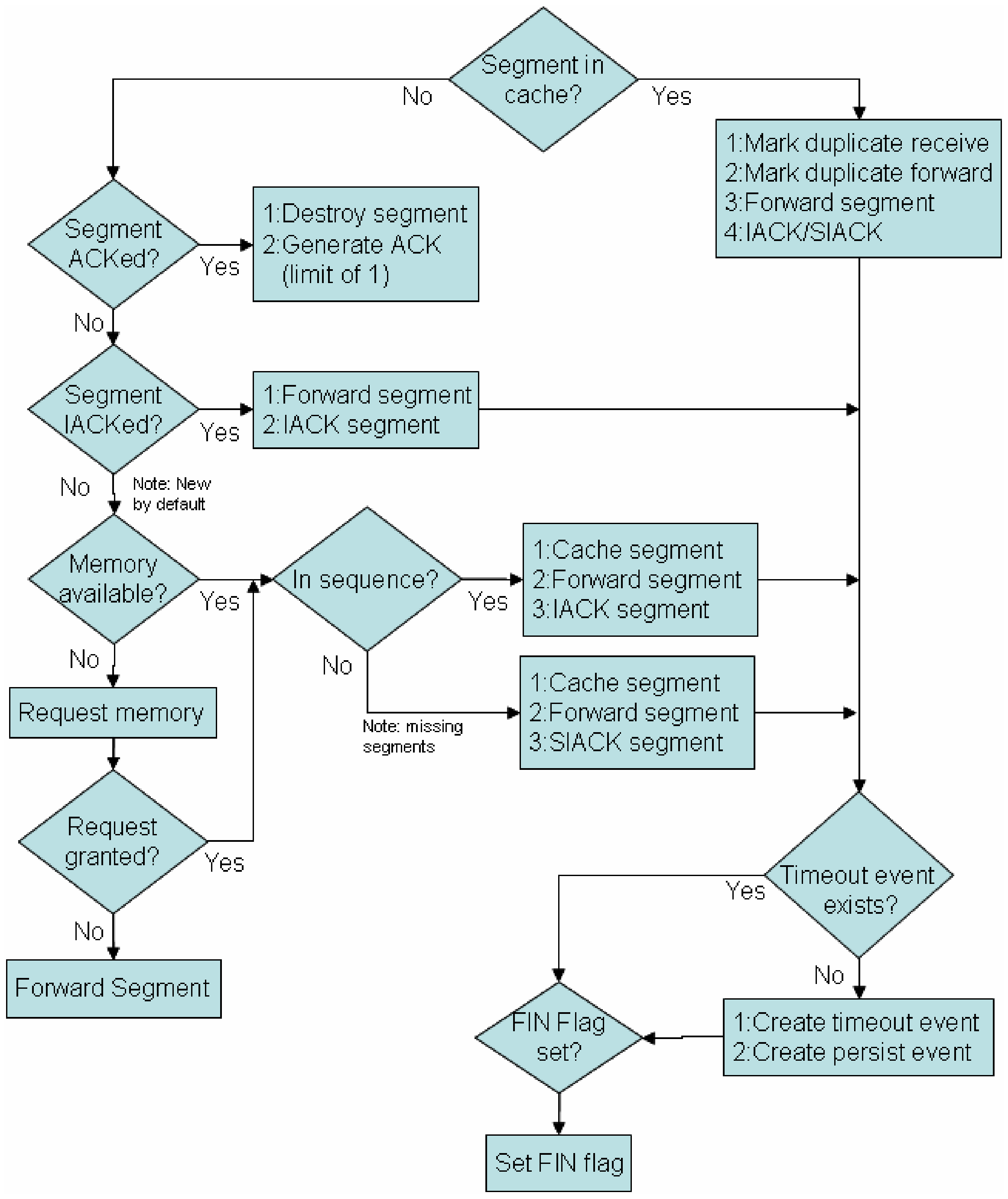


Figure 3.10: Proxy_handle_data Flow Chart

3.7.9 Proxy_handle_ACK Routine

Feedback datagrams are intercepted and used to clear cached segments and detect loss events. Feedback events consist of ACKs, IACKs, and persist response messages, each of which arrives for differing reasons, yet are used very similarly. A flow chart of actions performed in the proxy_handle_ack routine on datagrams meeting the Table 3.1 requirements is displayed in Figure 3.11.

All TCP segments transitioning through a proxy router may be intercepted by up to two **link proxies**, depending on internal routing. TCP can place acknowledgements for received data within its own data segments traveling to its communication partner. Thus, while transitioning through a proxy router, segments carrying data will be intercepted and handled by the proxy_handle_ACK routine on the incoming link and proxy_handle_data on the outbound link. The developed OPNET model focuses on FTP-like TCP traffic, thus acknowledgements will be received that should not be interpreted as duplicate acknowledgments since the receiver is not sending data. In such a case, only the required state data required for potential acknowledgement creation is recorded before the segment is forwarded for routing.

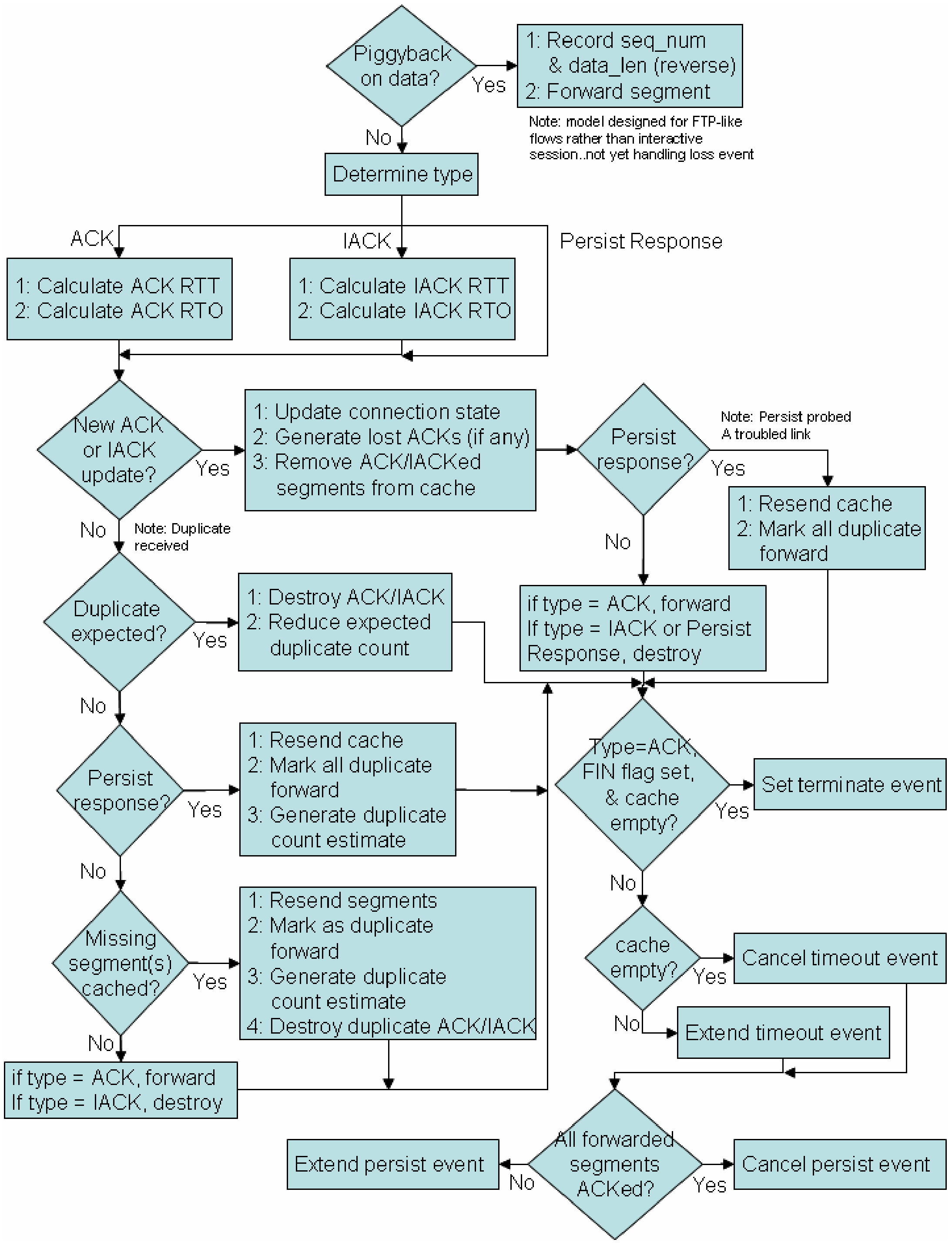
For the typical case of TCP acknowledgment only segment, or proxy IACK message, then the appropriate RTT and RTT variance calculations are performed. No such calculations are performed for proxy persist response messages. If the received feedback event carries new TCP acknowledgment or proxy IACK updates, then the new update is saved in the flow state data and acknowledged segments are removed from the **link proxy** buffer. If the update is for TCP acknowledgements, then perceived lost TCP acknowledgments are generated on behalf of the receiver and forwarded to the sender. New TCP acknowledgements are forwarded and proxy IACKs are destroyed.

If the feedback event is a proxy persist response message, then all flow segments remaining in the cache are immediately resent and marked as duplicate resends. These actions are necessary because a persist response is received only in direct response to a persist query message that probed the link for status. Any cached segment that is not acknowledged in the response was lost in transit between **link proxy** routers. Proxy persist responses are destroyed at the acting **link proxy**.

When the feedback event is a TCP acknowledgment segment or proxy IACK message that carries no new information and is thus a repeat, the **link proxy** interprets the repeat as a loss event. If the repeat message is expected by the **link proxy**, then internal state data is updated to reduce the expected repeat count and the repeat message is destroyed. If the repeat message was unexpected, then handling is dependent on the type of feedback event that was received. If the received message is a proxy persist response, all unacknowledged segments are resent immediately and state data is updated to note the resend and estimate of the number of repeat acknowledgements that could be seen.

Otherwise, the feedback event was a duplicate TCP acknowledgement or proxy IACK requiring a partial or complete resend of the flow cached segments. If lost segments are locally cached, then they are resent and marked as such in local state data. The feedback event message is then destroyed to avoid possible *congestion control* algorithm actions at the TCP sender. If lost segments are not locally cached, then only duplicate TCP acknowledgements are forwarded for action upstream.

Once all acknowledgement information has been gleaned from the feedback event, **link proxy** timeout event decisions must be made. If the TCP flow FIN flag was noted by proxy_handle_data and TCP acknowledgements have been received for all forwarded data, then a terminate event is scheduled for 60 seconds in the future to destroy the connection record. Not immediately destroying the connection record allows for handling any loss event messages that should arrive or TCP connection endpoint resends of flow termination messages. If the feedback event emptied the cache, then the retransmission timeout event needs to be cancelled. Otherwise, date remains in the cache and the retransmission timeout needs to be extended. Unless all forwarded segments have been acknowledged by the TCP receiver, a persist event needs to be scheduled to probe the link for the missing TCP acknowledgement information.



Piggyback on data?
Yes
1: Record seq_num & data_len (reverse)
2: Forward segment
Note: model designed for FTP-like flows rather than interactive session..not yet handling loss event
No
Determine type
ACK
IACK
Persist Response
1: Calculate ACK RTT
2: Calculate ACK RTO
1: Calculate IACK RTT
2: Calculate IACK RTO
New ACK or IACK update?
Yes
1: Update connection state
2: Generate lost ACKs (if any)
3: Remove ACK/IACKed segments from cache
Persist response?
Yes
Note: Persist probed A troubled link
1: Resend cache
2: Mark all duplicate forward
No
if type = ACK, forward
If type = IACK or Persist Response, destroy
No
Note: Duplicate received
Duplicate expected?
Yes
1: Destroy ACK/IACK
2: Reduce expected duplicate count
No
Persist response?
Yes
1: Resend cache
2: Mark all duplicate forward
3: Generate duplicate count estimate
No
Missing segment(s) cached?
Yes
1: Resend segments
2: Mark as duplicate forward
3: Generate duplicate count estimate
4: Destroy duplicate ACK/IACK
No
if type = ACK, forward
If type = IACK, destroy
Type=ACK, FIN flag set, & cache empty?
Yes
Set terminate event
No
cache empty?
Yes
Cancel timeout event
No
Extend timeout event
All forwarded segments ACKed?
No
Extend persist event
Yes
Cancel persist event

Figure 3.11: Proxy_handle_ACK Flow Chart

3.7.10 Round Trip Time Estimation and Timeout Calculation

Every **link proxy** maintains estimates for RTT and RTT variance to both the next **link proxy** and TCP destination for every flow it supports. This model employs the RFC 2988 [14] approach to calculating smoothed RTT and variance giving some weight to the most recent measurement but heavily weighting a historical trend. The alpha and beta values used in estimation are .125 and .25 respectively. RTT estimation is only performed when an acknowledgement is received with a matching cache entry. Additionally, if the cached segment is marked as resent, then the segment is not used for RTT calculation due to ambiguity (Karn's Algorithm [7]). Acknowledgement RTT and variance are used to calculate retransmission timeout (RTO) values for automatic resend events by summing the RTT and four times the variance. Including the variance component minimizes the potential for unnecessary resends. In the event of a timeout event, the RTO is doubled as it is in the TCP sender in order to minimize overwhelming the challenged link. The reader is reminded that a basic assumption of the model is that no link layer retransmission mechanism exists and thus all datagrams scheduled for transmission over a link are sent regardless of success or failure.

3.7.11 Event Timers and Timeout Events

Three types of timer-based events are maintained for each TCP flow in a **link proxy**. The timer types as well as initiation, extension, and cancellation of each are outlined in Table 3.2.

Table 3.2: Link Proxy Event timers

Timer type	**Initiation**	**Extension**	**Cancellation**
Retransmit	Data segment forwarded and no retransmission exists	Acknowledgment receipt and cache not empty	Cache empty
Persist	Data segment forwarded and no persist event exists	Acknowledgment Receipt and all forwarded segments not TCP acknowledged	All forwarded segments TCP acknowledged
Terminate	FIN set and cache empty	N/A	N/A

3.7.11.1 Retransmission Timer

A retransmission timer is used to retransmit all cached segments on a failed link. The retransmission timeout event is initiated when a data segment is forwarded and a retransmission timeout event does not yet exist. It is extended whenever an acknowledgement for the flow is received and the cache is not yet empty. The retransmission timeout event is cancelled when no segments are cached for a flow. The developed model uses the minimum of the calculated RTT values to calculate the retransmission event time, in effect using a developed knowledge of the **link proxy** position in the ETE flow. A 1 msec floor is placed on the retransmission timer to minimize the effects of small variance link.

3.7.11.2 Persist Timer

A persist timer is used to probe a questionable link when an expected outstanding acknowledgment has not been received from the TCP receiver. The model uses a default 10 msec persist timer rather than a tuned value based on acknowledgments from the receiver. The persist timer is initiated when a data segment is forwarded and a persist event does not exist. The timer is extended when a new acknowledgement is received and all segments have not been acknowledged by the TCP receiver. It is cancelled when all forwarded segments are TCP acknowledged. Note that an empty cache is not sufficient to cancel the persist timer as IACKs are not a substitute for true TCP acknowledgements. The TCP sender cannot interpret IACKs and thus requires a true TCP acknowledgement to advance the *send window*.

Persist timer expiration results in generation of a persist query that is forwarded downstream towards the TCP receiver. A persist query functions as a probe of the link and the downstream **link proxy** requesting an IACK and TCP receiver acknowledgement update. A received generated persist response will update the receiving **link proxy** with the most recent status of the nearest downstream **link proxy** and thus suggest the need for resending lost segments or generating lost acknowledgements.

3.7.11.3 Terminate Timer

The termination event timer is used to destroy a TCP flow connection record once the sender sets the FIN flag and all forwarded segments are acknowledged by the TCP receiver. The termination event timer is never extended or cancelled once set. The developed model does not currently generate a default terminate timer that will destroy connection records and unacknowledged segments in the event of a TCP sender unilaterally terminating a TCP connection without first setting a FIN flag.

3.7.12 Link Failure Detection

Loss events are discovered by **link proxies** via duplicate acknowledgements and event timer expirations. Duplicate acknowledgement handling was discussed previously, but event timer expiration requires further discussion. Upon expiration of a retransmission timer, all cached segments are automatically resent by a **link proxy**. It is entirely possible that all of the retransmitted segments may be lost or destroyed in the transmission. Each successive retransmission timeout results in an increased backoff of the retransmission timer.

Once all forwarded segments are acknowledged by IACKs, retransmission timeout events are no longer needed. However, it is possible that acknowledgements from the TCP receiver could be lost, resulting in a timeout event at the TCP sender. Thus, a persist event timer is used to probe the link and query for updated information from the downstream **link proxy**. Lost TCP receiver acknowledgements are discovered and regenerated as necessary via this method. Persist events also function as simple probes of a challenged link when the retransmission timer is in a wait period between successive retransmission events.

3.8 Memory Pool Manager

Each proxy router contains a central memory pool manager to coordinate shared memory pool usage of **link proxies** within the router. The inclusion of a memory pool manager allows future research expansion capability beyond this work. The OPNET process model for the memory pool manager is displayed in Figure 3.12 and each state is discussed in the following text.

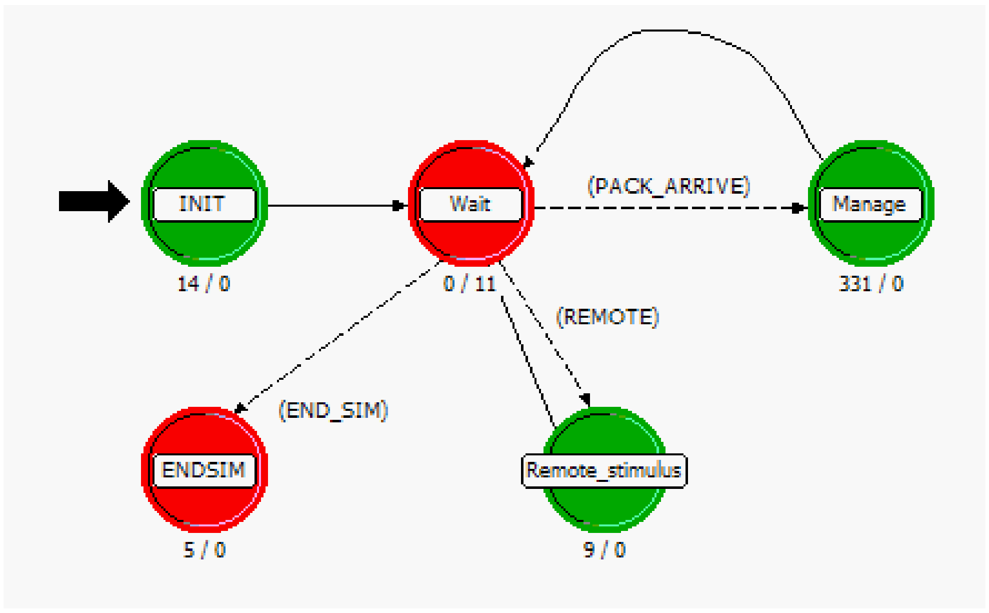


Figure 3.12: Memory Pool Manager Process Model

3.8.1 Init State

The Initialize state is initiated by the event simulation. Upon activation, the manager identifies itself and its parent process and reads its settable attributes for the simulation. The manager then initializes internal statistics variables and creates a list structure for storing link records. A shared memory allocation structure is initialized for each **link proxy** and a wakeup message with the link's shared memory pointer is sent to each for proper initialization.

3.8.2 Wait State

The wait state is a simple holding point for awaiting the arrival of a discrete event that requires attention in the memory pool manager. Interrupt types are classified as stream, remote, or end of simulation interrupts. Any interrupt to the memory pool manager process model must fall into one of the three categories and the process model passes to the Manage (stream), Remote_Stimulus(remote), or Endsim(endsim) states depending on interrupt type.

3.8.3 Manage State

Communication between OPNET object models occurs primarily with communication packets between the objects. The manage state receives all requests from **link proxies** and handles generation of the appropriate response. Incoming requests handled include memory allocation requests and connection termination notices. The developed model simply supports memory allocation requests in blocks of ten packets up to the user specified available memory limit. Requests are supported on a first come first served basis and no connection priority scheme currently exists.

3.8.4 Remote Stimulus State

Whenever a **link proxy** uses or clears cache memory, it generates a remote stimulus event allowing the memory manager to perform reallocation algorithms. The remote stimulus state handles the memory reallocation procedure and updates local state information necessary for maintaining prioritized flow performance. The current model does not perform any reallocation actions, however the model is easily modifiable for anticipated follow-on research activities.

3.8.5 Endsim State

The Endsim state is activated at the completion of a simulation and serves as a state to handle graceful memory recovery actions.

3.9 Link Failure Model

A challenged link is modeled via evaluating link status on a constant interval as designated by the user. The probability of a link failing is user settable. A link is either up for the interval period or it is down. Every wireless link performs an independent status evaluation at the beginning of an interval. The developed model allows datagrams to transit a challenged link unhindered when the link is up. The model also allows datagrams to be placed on a failed link and destroyed "in transit" when the link is down.

3.10 Summary

Chapter Three presented the proxy router and **link proxy** concepts used to overcome a challenged link environment. The developed model is presented in detail from the router level down to the individual **link proxy** level. The challenged link model utilized in this research effort is also presented. Chapter Four introduces the network simulation methodology and results of the research effort.

IV. Evaluation Methodology, Analysis, and Results

4.1 Chapter Overview

The methodology for evaluating the model presented in Chapter Three is presented in this chapter. The investigative questions are introduced along with a context of the entire system under test and selected parameters influencing the System Under Test (SUT). The experimental design for each investigative question is introduced followed by an analysis of the obtained results.

4.2 Methodology

This research is a combination of proof of concept and comparative analysis of improvement over an existing TCP protocol. The focused goal is to improve throughput for FTP-like TCP data stream transmissions in a challenged network. These streams consist of a single ETE TCP connection over which large amounts of data are sent. An additional goal is to ensure neither fair access to the network nor throughput for other TCP byte streams suffer as a result of the changes introduced. The hypothesis is that introducing **link proxies** on each challenged link in a multi-hop network will improve network throughput and reduce ETE delay for TCP streams.

Specific questions this study will answer include:

1- Can implementing **link proxy** routers hide non-congestion losses from a TCP sender without modifications to the TCP endpoints?

2- Does implementing **link proxy** routers improve throughput in a challenged environment when compared to TCP Reno?

3- Does implementing **link proxy** routers improve throughput in a multi-hop challenged environment?

4- Does implementing **link proxy** routers negatively affect non-modified routers within the network?

5- Does supporting TCP byte streams with **link proxy** routers negatively affect fairness of other TCP byte streams within the network?

4.2.1 System Boundaries

The introduction of **link proxy** routers allows the network to support TCP ETE communication without altering ETE semantics of the TCP protocol or the implementation of the protocol at the TCP communication pair. Therefore, the system under study consists of a defined network topology along with the physical equipment and protocols. Also included within the system boundary is the environment, or "weather" within the network, including wireless link characteristics and all traffic within the network. Primary limitations of the system under study are a static topology and stationary nodes. Accordingly, routing algorithms are outside of the scope of the SUT.

4.2.2 System Services

The System Under Test exists for the sole purpose of transporting data packets between sources and destinations. A key design goal is for packets to be transported in the most efficient and reliable manner possible, with minimum ETE delay. Once a packet enters the system, it is either delivered to a host or not. A packet is not delivered if it is dropped by the network as a result of either a buffer overflow, a bad checksum (due to transmission error), or time to live (TTL) counter expiration. The packet could also simply be lost due to router malfunction. If a packet is delivered to a host, it is either the correct host or not. If the packet arrives at the correct host, it is considered successfully delivered as long as the checksum is correct.

4.2.3 Workload

The workload for a network is the amount of traffic submitted to the network for delivery. An individual packet travels from a source to a specific destination, yet the network also carries other traffic for which resource contention is an issue. Differing protocols can be used for packet transmission as the packet traverses through the network.

For this study, all traffic within the network is synthetic FTP traffic where files are transferred between source/destination pairs using the network layer TCP protocol. Each file transfer consists of many packets that comprise segments of a TCP byte stream. The number of individual packets and their size change, but a given TCP connection is responsible for moving a total aggregate amount of data between source and destination pairs.

Note the network load for this study consists of only the synthetic connection establishment and FTP traffic specifically noted in the tested scenarios. No background load exists within the network. Accordingly, conclusions derived in this thesis are suggestive only and larger network simulations with increased background load should be performed by any follow-on activity.

4.2.4 Performance Metrics

Table 4.1 displays the metrics of interest in this study. Each metric, its definition, and a description of the metric's use are outlined in the table. Since the system under test is relatively large in scope, various network layer metrics are used to analyze the system. Some of the metrics are derived metrics.

Table 4.1: Performance Metrics

Metric	**Definition**	**Use**	**Units**
Link Capacity	The maximum (analytical) rate at which data can traverse adjacent nodes	Theoretical maximum amount of data that can flow across a link	Bits per second
Link Throughput	The measured rate at which data traverses adjacent nodes	Measured amount of data flow between adjacent nodes	Bits per second
Link Utilization	The ratio of link throughput to link capacity	Measure of efficiency of the link	None
Mean delay (packet)	Mean delay for a packet sent between a source/destination pair measured from the time the first bit of the packet is sent to the time the last bit is received	Measure of one way traversal time for a packet to traverse the network	Seconds
Mean delay (byte stream)	Mean delay for the complete TCP byte stream between a source/destination pair measured from the time the first bit of the first packet is sent to the time the last bit of the final packet is received	Measure of delay for a file to traverse the network	Seconds
Total bytes sent (byte stream)	The aggregate number of sent bytes (including overhead) for a specific TCP source/destination byte stream	Raw data for rate determination.	Bytes
Total bytes delivered (byte stream)	The aggregate number of bytes delivered (including overhead) for a specific TCP source/destination byte stream	Raw data for rate determination	Bytes

4.2.5 Parameters

Many parameters affect performance of the system under test. These parameters are classified as system parameters if they directly contribute to network performance or as workload parameters if they are characteristics of the workload. For ease of review, the parameters are displayed in Table 4.2 along with their effect on the system.

Table 4.2: System Under Test Parameters

Parameter	**Units**	**System / Workload**	**Main Effect**
Number of nodes	Nodes	System	Packet delay from router processing (queuing delay)
Distance between adjacent nodes	Meters	System	Packet propagation delay
Transmission medium	N/A	System	Resource contention (wireless shared/wired dedicated) and specific physical layer protocols
Bit Error Ratio (BER)	None	System	Link layer error rate due to channel effects
Link capacity	Bits per second	System	Upper bound on traffic carrying capacity between adjacent nodes
Service rate of a router	Bits per second	System	Transmission delay at a node
Buffer size	Bytes	System	Amount of data that can be queued without dropping packets
Packet Size (maximum)	Bytes	System	Influences upper bound on transmission delay and affects queue capacity
Window size	Packets	System	Number of unacknowledged packets in transmission at a time between sender and receiver
Congestion control scheme	N/A	System	Sender side backoff mechanism strategy to reduce packet input rate to network
Resource utilization	N/A (ratio)	System	Influences queuing delay and dropped packet probability
Packet acknowledgment	N/A	System	(Cumulative, Selective, Intermediate) Directly impacts number of acknowledgments required and potential for wasted capacity due to unnecessary resends

Transport, Network, Link, and Physical layer protocols	N/A	System	Influences transmission time and reliability of transmission
File size	Bytes	Workload	ETE delay of full transmission
Network offered load	N/A (ratio)	Workload	Resource contention affects transmission time, congestion, fairness

4.2.6 Factors

Factors selected for investigation during this study are summarized in Table 4.3. The primary focus of this research is implementing a **link proxy** capable router adjacent to each challenged link in the network and evaluating the throughput differences from a standard router. The level at which wireless connections are challenged is modeled as a combination of the link failure probability and loss interval period to determine the degree of improvement at varying degrees of challenge.

Table 4.3: Experiment Factors

Factor	**Levels**
Link Proxy Capability Status	1) Enabled 2) Disabled
Link Failure Probability	1-9) 0% - 40% in 5% increments
Loss Interval Period	1-54) 40 – 1100 msec in 20 msec increments

4.2.7 Experimental Design

The strategy for completing this study is to examine the effects of each factor on system performance with a full factorial experiment using the factors highlighted in Table 4-3. A standard static network configuration is implemented with **link proxy** routers enabled or disabled. Using each **link proxy** configuration, nine levels of link failure probability are simulated with fifty-four levels of loss interval period. Without replications, this requires 972 experiments for each network configuration. Thirty replications, using unique seeds, are used for each experiment to reduce the variance of the collected data.

The magnitude of data collection for each experiment is significant. In order to answer the study goals, ETE delay statistics are collected for every packet and complete TCP byte stream of every source/destination tuplet. Additionally, statistics are collected for every link and node within the network to provide insight into the study goals.

4.2.8 Experimental Parameter Settings

Three simulation scenarios were investigated for this study to answer the questions raised earlier in this chapter. All simulations were performed using the OPNET simulation tool and the models presented in Chapter Three. All links in the model are 100Mbit PPP links with speed of light propagation and zero inherent BER. The maximum transmission unit for the PPP links is set at 1500 bytes. Challenged links contain the link failure model (packet discarder) introduced in Chapter Three and introduce no additional delay for packets traversing the link during a non-loss period.

All routers used in the simulations are **link proxy** routers where each **link proxy** can be independently enabled or disabled. Disabled link proxies simply pass all packets through the proxy unhindered and without inspection. The routers are central-CPU based with a service rate of 50,000 packets per second and infinite queues. Static routing is utilized to remove overhead traffic associated with routing protocols. The router settings ensure that there are no congestion losses in the network and only loss events introduced by challenged links affect the TCP sender.

Each **link proxy** router is configured with 1.5MB of centrally managed memory for **link proxy** use. Link proxies are modeled with no delay. It is fully understood that claiming zero delay is somewhat unreasonable, but this study focused on the proof of concept rather than full implementation details.

ETE communication is modeled using TCP Reno [15], which introduced *congestion control* to TCP along with loss event *fast retransmit* and *fast recovery* mechanisms. All TCP *receive window* buffers are 65535 bytes. General TCP implementations use a cumulative acknowledgment scheme with a maximum acknowledgment delay of 200 msec or two segments. During *slow start*, a TCP Reno receiver will wait 200 msec before responding to the very first segment from the sender. A **link proxy** router however will interpret such a long delay as channel loss, resending the segment in question. The arrival of the resent segment meets the two segment maximum delay requirement, forcing a receiver acknowledgment and providing an almost 200 msec performance gain for **link proxy** routers. Accordingly, to avoid an unfair evaluation advantage (artificial gain due to TCP endpoint settings) for **link proxy** routers, a maximum delay of 1 msec or two segments is utilized.

The *slow start* initial *congestion window* is 1500 bytes and the three duplicate acknowledgments invoke *congestion control*. Karn's algorithm is followed, discounting resent segments in ETE RTT calculations. Initial retransmission timeout is set at 1 sec and varies between a minimum 0.5 sec and maximum 64 sec value as computed by the TCP sender. A maximum of three connection attempts is allowed by a TCP connection initiator during connection establishment. Once a connection has been established, 6 back-to back data retransmissions timeouts are allowed by a TCP sender before a connection is terminated.

It should be noted that the majority of these TCP parameters are user selectable and several other "flavors" of TCP exist. The results of this study may not be directly attributable to other forms of TCP and direct comparison to other TCP parameter settings should be exercised with caution.

4.3 Investigative Questions Answered

4.3.1 Question 1

Can implementing **link proxy** routers hide non-congestion losses from a TCP sender without modifications to the TCP endpoints? In order to answer this question, the topology displayed in Figure 4.1 is simulated using the factors described in Table 4.3. The scenario consists of a client initiating a TCP connection at simulation time 15 seconds and requesting an FTP transfer of a 20MB file from a remote server. Two **link proxy** routers are used to support a single challenged link in a three-hop path between the client and server. Execution for this scenario consists of performing 29,160 runs using an OPNET command line execution input generated by a script batch file. The script specifies the unique scenario settings to include seed value, probability of link failure, and link failure interval.

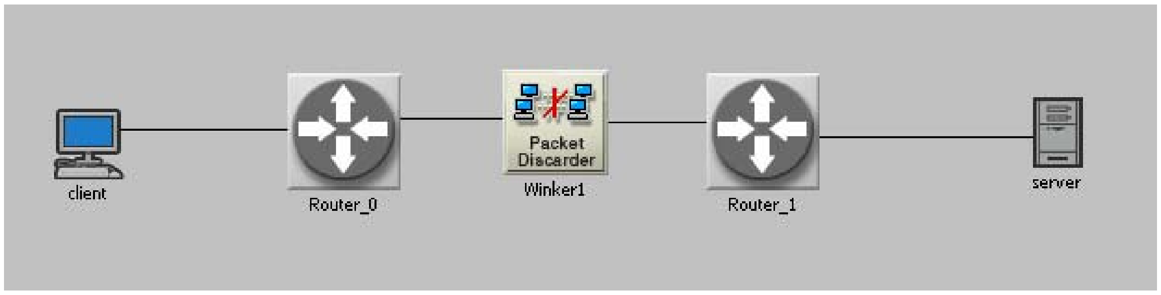


Figure 4.1: Single challenged link topology

As discussed in Chapter Two, challenged links introduce two primary complications to TCP connections. First, connection establishment requires a three-way handshake between an initiator and receiver. When links are non-challenged, the datagrams carrying the connection establishment segments are easily received. In a heavy loss environment however, connection initiation segments can be easily lost, potentially resulting in a connection establishment delay or even worse, an abort. Once a connection is established, data segments and acknowledgments must both traverse the challenged link. In a heavy loss environment, many of both types of segments are lost. The combination of lost data segments (and the resulting lack of receiver generated acknowledgments) and loss of generated acknowledgments can cause the TCP sender to abort the connection once six back-to-back retransmission timeout events have occurred.

Figure 4.2 displays the number of TCP aborts recorded for the scenario when the challenged link was not supported with **link proxy** routers. Every abort is a result of the failure of the communicating endpoints to establish a connection and initiate an FTP transfer. The z-axis is the total number of TCP aborts for the 30 independent runs for each probability of failure and failure interval pairing. The x-axis is the failure interval and the y-axis is the probability of failure of the challenged link.

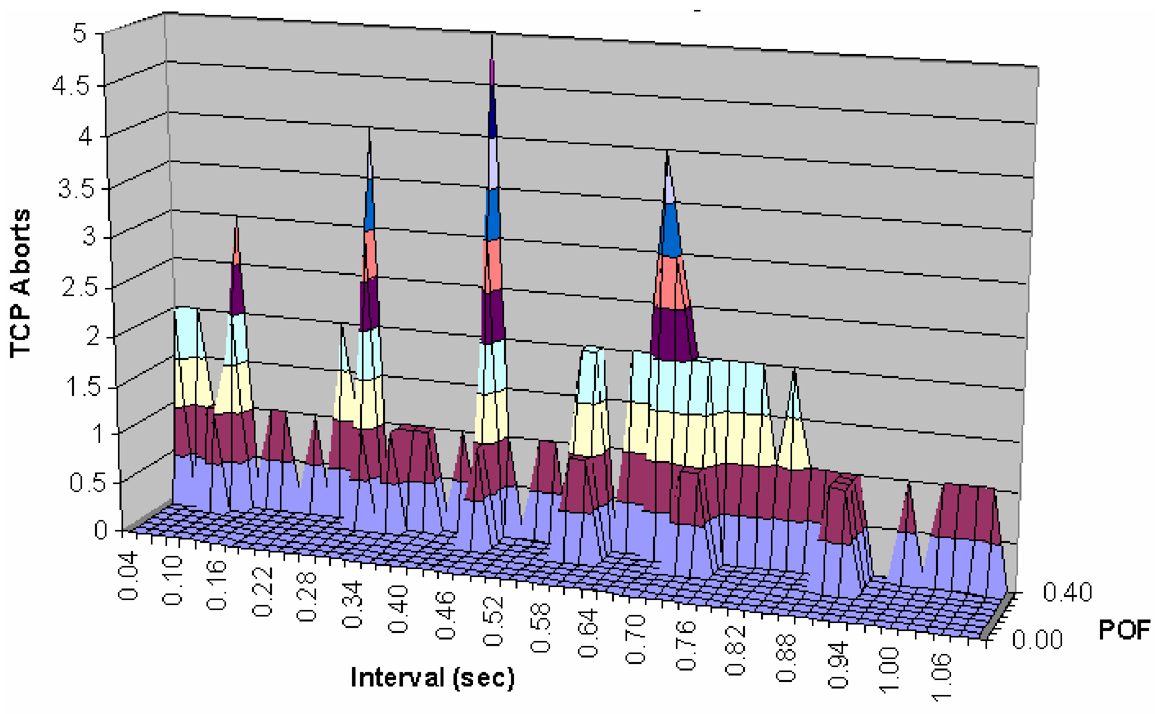


Figure 4.2: TCP Aborts Observed for Single Challenged Link using Standard Router Configuration

Connection aborts occur for link probabilities of 25% and greater. The greatest number of failures are observed at 40% probability of failure and 460 msec link failure interval. 9% of the failures are observed at the 460 msec interval and another 35% of the total failures are observed between 580 and 740 msec intervals. 56% of the failures are attributed to five seed values which appear to stress the connection establishment process. While no direct conclusions are drawn with respect to the differing interval periods, there does appear to be a difference in TCP behavior depending on the timescale of the dropout periods.

In stark contrast, when proxy routers are utilized on the challenged link, there are absolutely no TCP abort events for the factor test points, as a direct result of the buffering strategy implementation design decision to buffer all TCP traffic, regardless of ETE connection state. TCP connection establishment segments are buffered and treated like established connection data segments, allowing TCP communication support, even during connection establishment.

For non-aborted FTP transfers, a TCP sender's *congestion window* behavior, with and without use of **link proxy** routers, is as shown in Figure 4.3. The y-axis represents the *congestion window*, in bytes, of the TCP sender and the x-axis represents simulation time. Recall that the TCP sender allows the amount of unacknowledged data in the network to be the minimum of the receiver's advertised window, which is generally 65535 bytes, and the *congestion window*. The *congestion window* increases when an acknowledgment is received, however the amount of increase is dictated by the mode (*slow start* or *congestion avoidance*) in which TCP is operating. Three duplicate acknowledgments result in 50% reduction in the *congestion window* (not shown) and a sender retransmission timeout results in the *congestion window* resetting to 1460 bytes.

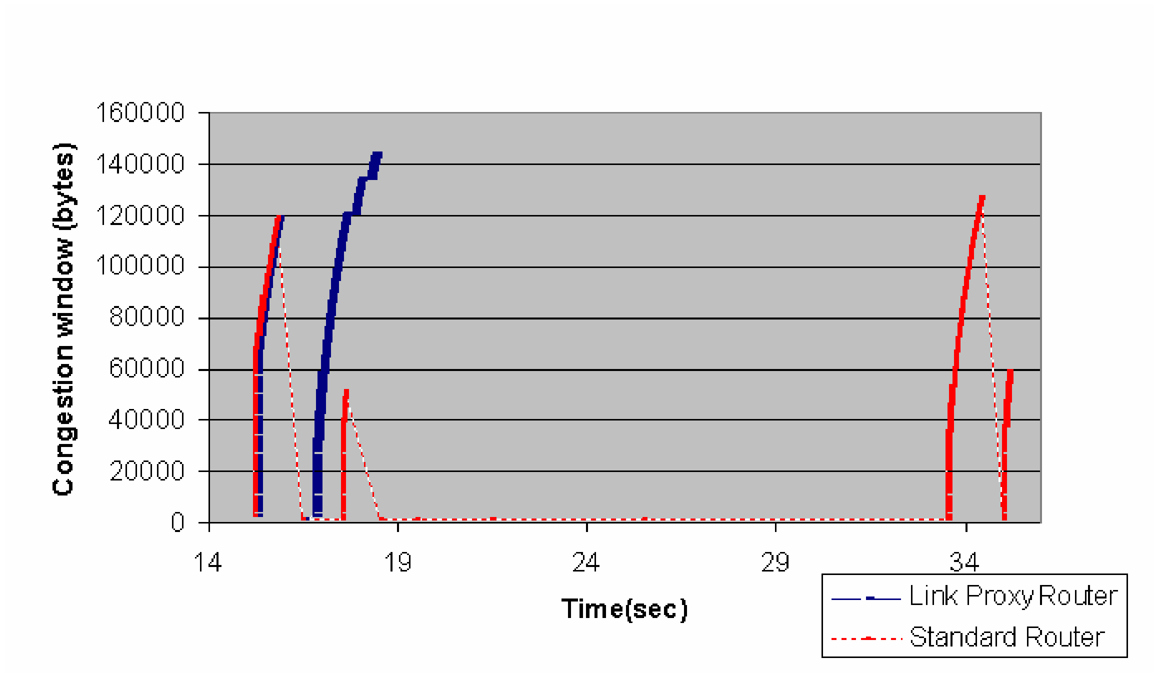


Figure 4.3: TCP Sender Congestion Window Behavior for Single Challenged Link with 20% Probability of Failure and 200 msec Failure Interval

Using standard routers, acknowledgments for outstanding transmitted data are not arriving at the TCP sender, causing retransmission timeout events. Acknowledgments are failing to arrive because datagrams carrying data to the receiver and acknowledgments from the receiver are lost. After a retransmission timeout, the TCP sender re-enters *slow start* and increases the amount of allowed outstanding segments with every received acknowledgment. In this particular example, eight retransmission timeouts occur with three of the timeout recoveries reentering the *congestion avoidance* phase. The other five retransmission timeouts fail to receive any acknowledgments and the retransmission timer backs off exponentially. The combination of time spent waiting for retransmission timeouts and slowly rebuilding the *congestion window* severely hampers TCP ETE throughput performance.

TCP sender *congestion window* performance remains nearly identical when the ETE path between the sender and receiver is unchallenged. Minute differences exist as a result of additional queuing delay from the slight overhead increase of intermediate acknowledgements, however theses differences are indistinguishable. This point is displayed in Figure 4.3 for the initial *congestion window* climb from 1460 to 120000 bytes.

By design, proxy routers do not insulate the TCP sender from genuine challenged link blackout periods exceeding the calculated TCP sender retransmission timeout timer. For this particular example, five consecutive failure periods resulted in a total link failure of one second, surpassing the TCP sender calculated retransmission timer and invoking *slow start congestion avoidance* and its corresponding *congestion window* reset to 1460 bytes.

When a proxy router is utilized, TCP segments are temporarily buffered at the router adjoining the challenged link. By maintaining round trip time estimates to the next router, using intermediate acknowledgments, and with persist queries, the **link proxy** discovers the loss events immediately and resend lost segments at the point of loss as soon as the link is discovered operational. Lost acknowledgments are also regenerated on behalf of the TCP receiver, allowing the TCP sender to increase the *congestion window* quickly. Maintaining the *congestion window* in excess of the *receive window* ensures that the maximum amount of outstanding data, as controlled by the receiver, remains in the network for delivery. In effect, the TCP sender has been insulated from challenged link non-congestion loss events, avoiding the invocation of congestion response handling and an immediate drop in TCP ETE throughput.

Figure 4.4 displays the accompanying TCP segment arrivals at the receiver and outbound **link proxy** buffer usage at Router 1. The gaps in segment arrival at the receiver are due to the single challenged link failure. When the challenged link is not experiencing loss, buffer usage is generally limited to one or two segments. When the challenged link fails, all segments traveling over the link, including proxy intermediate acknowledgments and persist queries/responses, are lost. **Link proxy** cache buffer usage climbs to 44 segments (64240 bytes) when the link fails and quickly drops back to minimal use once the link is discovered to be functional with receipt of intermediate or receiver acknowledgments. The peak usage matches the obligation of the TCP sender not to allow more outstanding data in the network than the TCP receiver makes allowance for in the *receive window* advertisement.

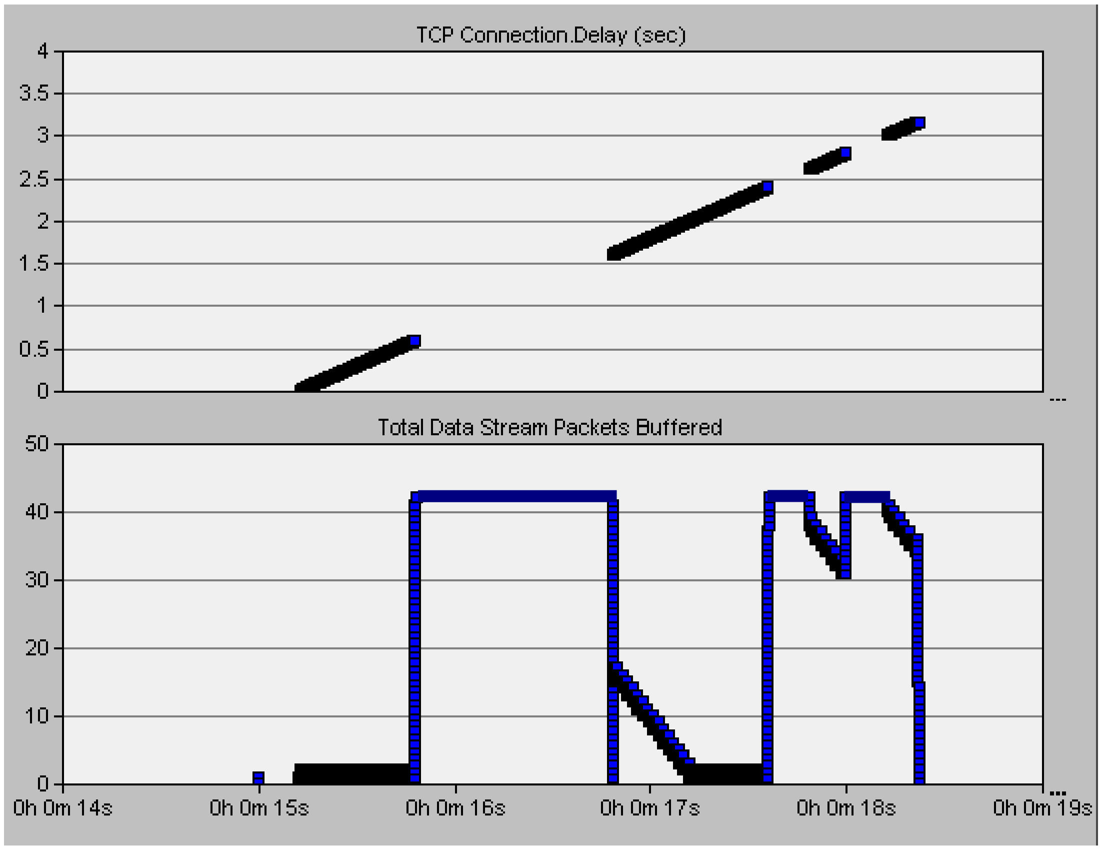


Figure 4.4: TCP Receiver Segment Arrival and Accompanying Router 1 Link Proxy Buffer Usage for Single Challenged Link with 20% Probability of Failure and 200 msec Failure Interval (Seed 137)

4.3.2 Question 2

Does implementing **link proxy** routers improve throughput in a challenged environment when compared to TCP Reno? Using the data collected answering question one, it is possible to determine if ETE TCP throughput is increased using **link proxy** routers. The average transfer time for non-aborted FTP transfers, using standard routers, is displayed in Figure 4.5 and the variance associated with the data is displayed in Figure 4.6. Note that the average FTP transfer time does not account for the impact of delayed connection establishment resulting from challenged link failure periods. The z-axis is the average transmission time for the successful FTP attempts (30 independent runs) for each probability of failure and failure interval pairing. The x-axis is the failure interval and the y-axis is the probability of failure of the challenged link. Under perfect conditions, transmission of the 20MB file, defined as the total time taken from the time the first byte is received to the last byte is received, requires 1.734 seconds.

The primary trend is the shorter the link failure interval, the greater the impact to ETE TCP communication. Increasing the probability of link failure also increases the communication time. FTP TCP traffic tends to burst as each round of transmission by the TCP sender consists of several segments sent back to back. Acknowledgments from the receiver also tend to cluster as the data arrives in rapid succession followed by a brief respite before the next transmission round. In the scenario under consideration, ETE delay for a segment is approximately 5.25 msec. Accordingly, failure intervals from 40 – 1100 msec will result in the loss of essentially all data and acknowledgment segments in a transmission round. Decreasing the loss interval period also increases the number of failure periods for consideration.

The increase in communication time from the baseline 1.734 seconds is primarily attributed to three factors. First, increasing the probability of failure reduces the challenged link capacity available for use. Second, the TCP sender spends a significant amount of time waiting for a timeout event to occur and thus take note that a loss has occurred. Third, higher probabilities of failure of the link increase the risk of back to back losses, causing the TCP sender to exponentially increase the backoff time between successive retransmission timeout events.

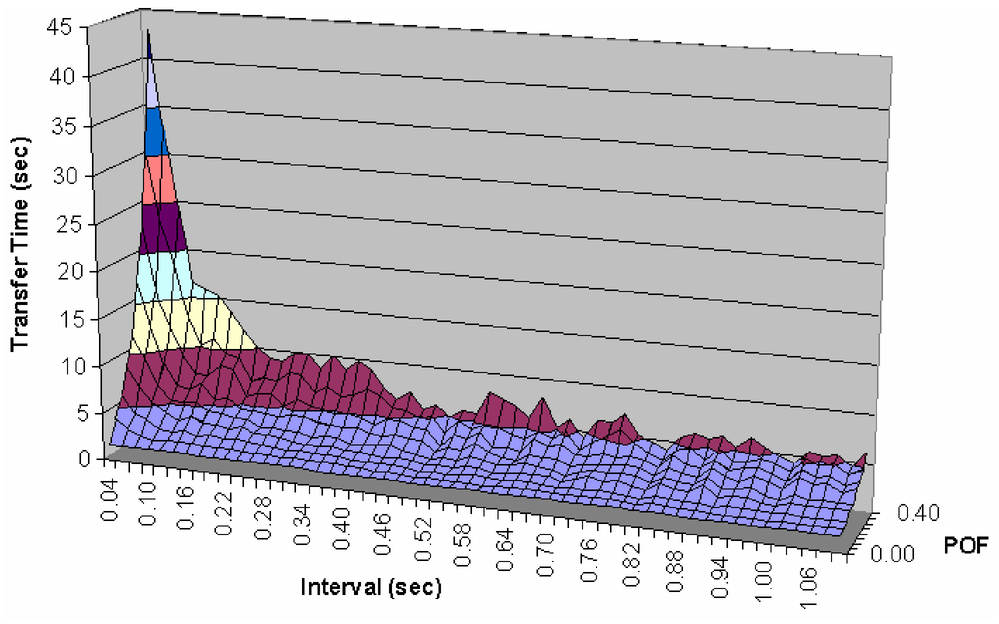


Figure 4.5: Average FTP Transfer Time of 20MB File over Single Challenged Link with Standard Routers

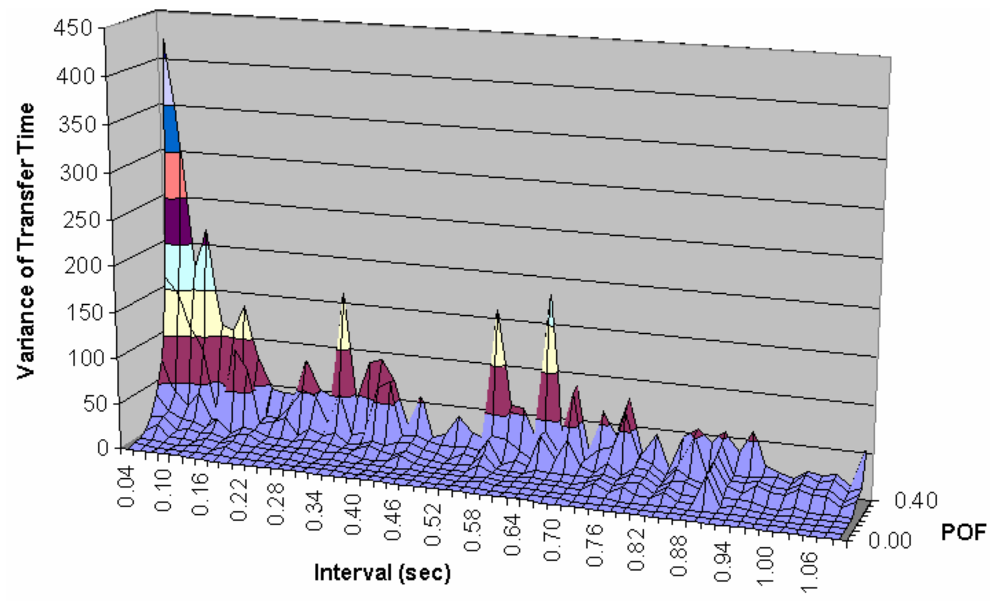


Figure 4.6: Variance of Average FTP Transfer Time of 20MB File over Single Challenged Link with Standard Routers

The valleys noted in the transfer time results of Figure 4.5 are the result of TCP timer settings used in the scenario. The minimum retransmission timeout timer value is set at 500 msec and multiples of 500 msec (500 msec and 1000 msec) show dips in FTP transmission time. Curious dips are noted at 720 and 820 msec and appear to be a harmonic effect of the 500 msec timer.

The variance of the transfer time data spiked at 400 for 40% probability of failure and 40 msec failure interval. Several of the probabilities of failure and interval combination variance values are greater than 100, suggesting that more data samples should be collected for a higher confidence in the mean values, especially for intervals less than 200 msec.

In contrast, the mean transfer time for FTP transfers using **link proxy** routers is displayed in Figure 4.7 and the variance associated with the data is displayed in Figure 4.8. Under perfect conditions, transmission of the 20MB file requires 1.734 seconds. The longest mean transfer time noted was 3.356 seconds (versus 43.1 seconds using standard routers), recorded for 40% probability of failure and 100 msec failure interval.

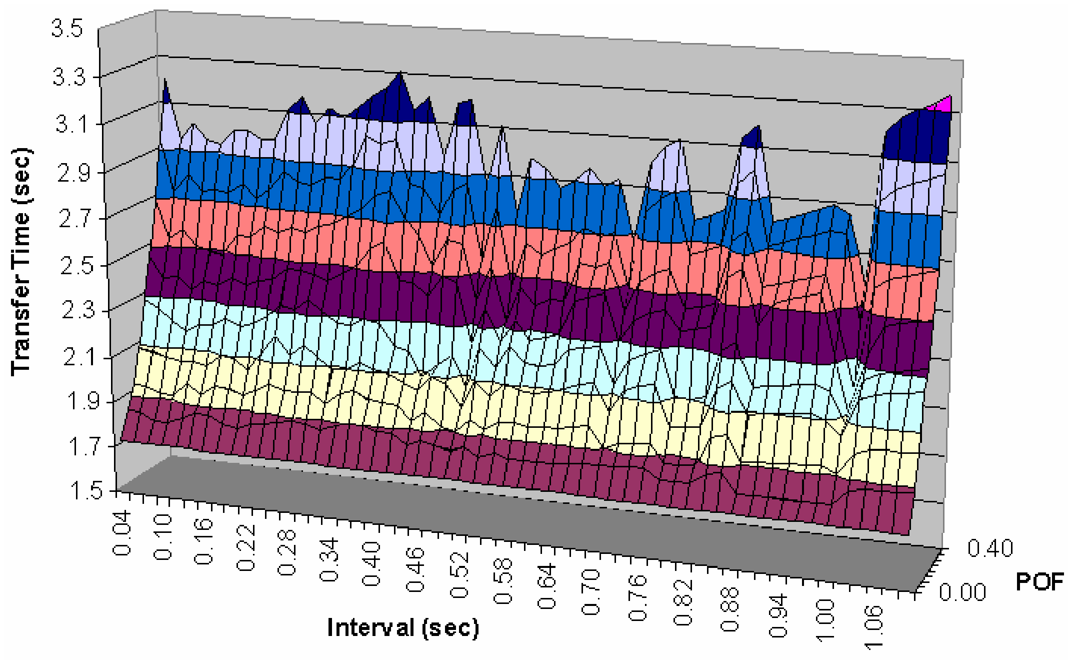


Figure 4.7: Average FTP Transfer Time of 20MB File over Single Challenged Link with Link Proxy Routers

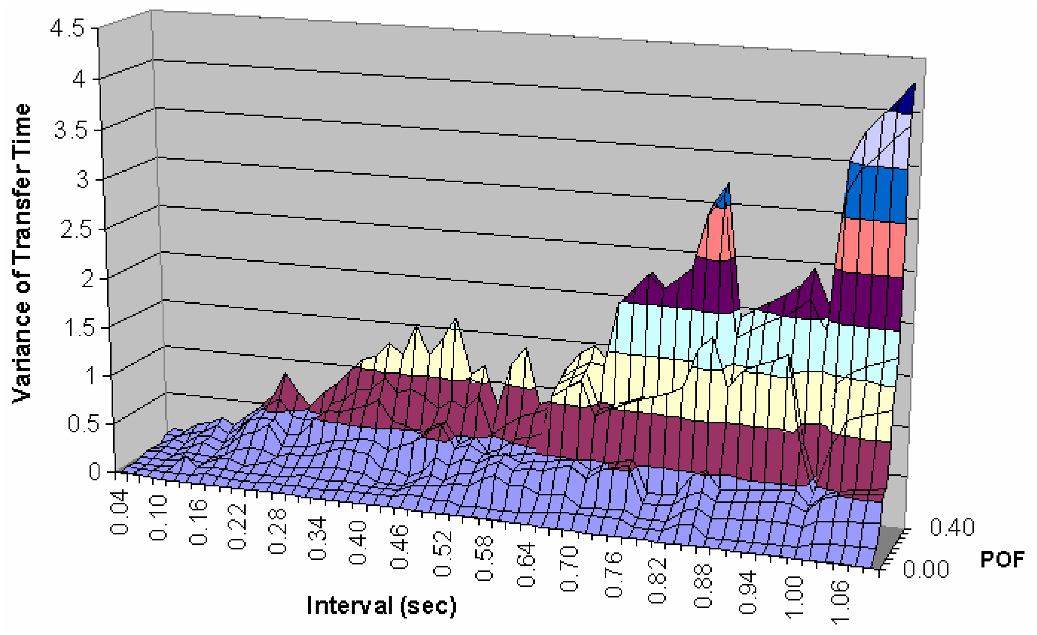


Figure 4.8: Variance of Average FTP Transfer Time of 20MB File over Single Challenged Link with Link Proxy Routers

The limited increase in communication time from the baseline 1.734 seconds is primarily attributed to a **link proxy's** ability to quickly discover that the challenged link is available for use and retransmit lost segments. **Link proxy** faster discovery and recovery of loss events keep the TCP sender from unwittingly waiting for timeout events to occur before loss discovery. Additionally, avoiding TCP sender timeout events reduces the probability (and impact) of back to back timeout events incurring the exponential backoff *congestion control* mechanism between successive retransmission timeout events.

The valleys noted in the transfer time results of standard routers are again noted when utilizing **link proxy** routers. This reinforces the idea that the valleys are the result of TCP timer settings. Of particular note, the degree of impact of the timers is significantly reduced from an order of seconds to milliseconds. Though reduced, the impact is still present because **link proxy** routers do not fully insulate a TCP sender from persistent link failures exceeding the sender's retransmission timeout timer.

The variance of the transfer time data spiked at 4.26 for 40% probability of failure and 1100 msec failure interval. The low variance of the observed data suggests that the results are well behaved and additional samples are not required. The increase in variance, particularly for the larger intervals, is a result of the small total file transfer size relative to the link outage duration. For the 1100 msec failure interval, only two link availability opportunities are required for successful file transmission. However, the reduced probability of having those opportunities results in a wider range of observed file transfer times.

The overall improvement observed for each probability of failure and link failure interval pair is displayed in Figure 4.9. The most dramatic impact is noted for link failure intervals less than 500 msec and increased link probability of failure rates. These results are inline with the previously discussed findings that a **link proxy** can quickly recover from short duration outages and recover faster than a TCP sender alone would even discover that segments are lost. Quicker discovery of the available bandwidth is immediately seized upon and used. Though not as dramatic, an improvement still exists for timeout intervals greater than 500 msec for the same reason.

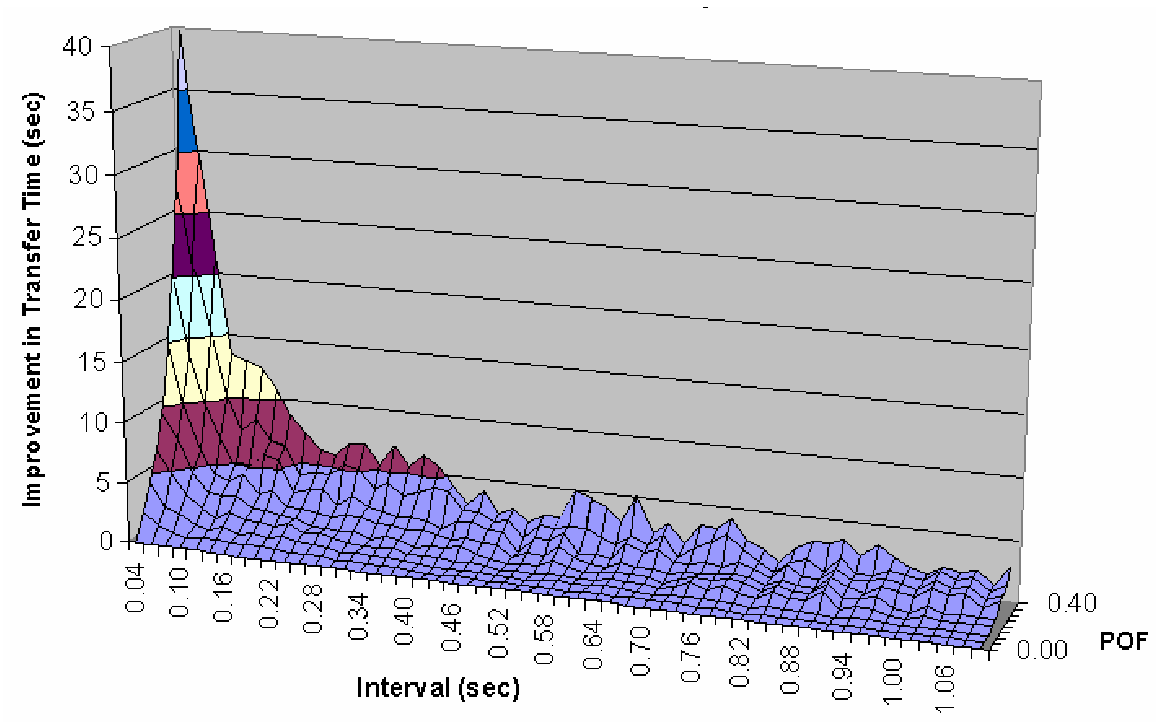


Figure 4.9: Improvement in Mean FTP Transfer Time of 20MB File over Single Challenged Link through Use of Link Proxy Routers

Mean ETE TCP throughput, defined as the amount of data transferred divided by the mean transfer time, is calculated for each probability of failure, link failure interval, and router type pairing. Available ETE bandwidth is reduced as a result of the challenged link bottleneck. The difference in utilized ETE bandwidth, adjusted for the challenged link expected available bandwidth bottleneck is displayed in Figure 4.10. As expected, using **link proxy** routers enables increased bandwidth usage across the board due to near-immediate loss event detection. The most notable gains were again noted for intervals less than 500 msec with a downward trend as the interval nears 500 msec. Dips in performance gains are again found at 500, 700, and 840 msec, corresponding to the TCP sender timer settings.

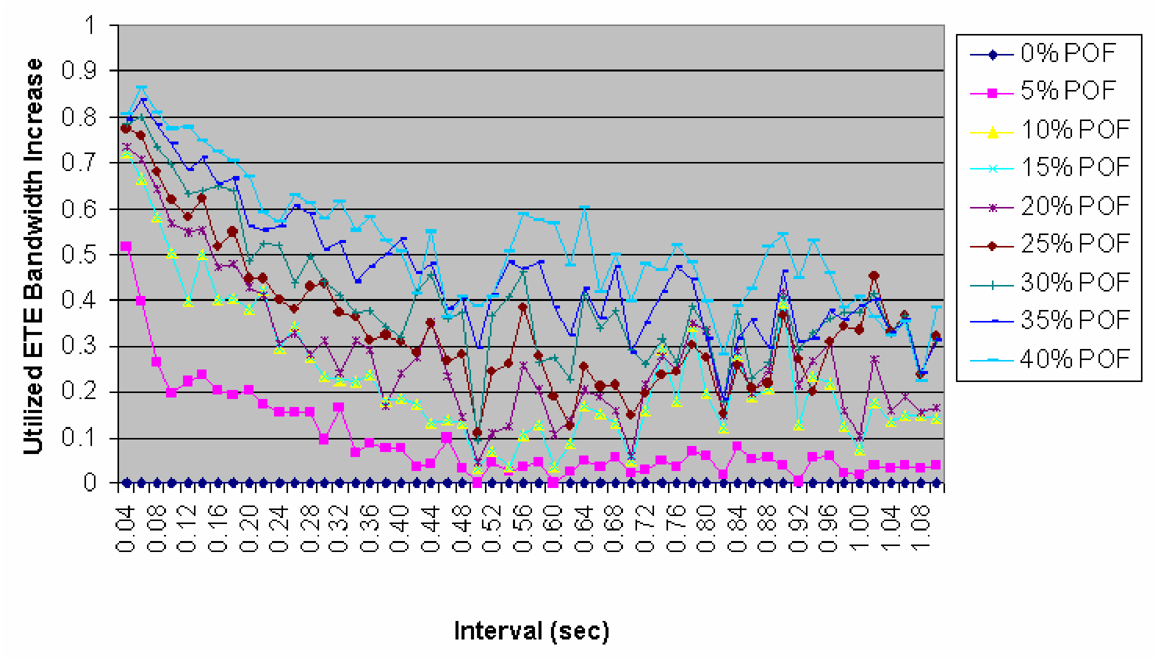


Figure 4.10: Improvement in Utilized ETE Bandwidth over Single Challenged Link Using Link Proxy Routers

4.3.3 Question 3

Does implementing **link proxy** routers improve throughput in a multi-hop challenged environment? The topology displayed in Figure 4.11 is simulated using the factors described in Table 4.3 to answer this question. The scenario consists of a client initiating a TCP connection at simulation time 15 seconds and requesting an FTP transfer of a 20MB file from a remote server. Five **link proxy** routers are used to support four challenged links in a six-hop path between the client and server. ETE connectivity follows the four challenged link curve plotted in Figure 1.1 and ranges from full connectivity to 13% connectivity. Execution for this scenario consists of performing 29,160 runs using an OPNET command line execution input generated by a script batch file. The script specifies the unique scenario settings to include seed value, probability of link failure, and link failure interval.

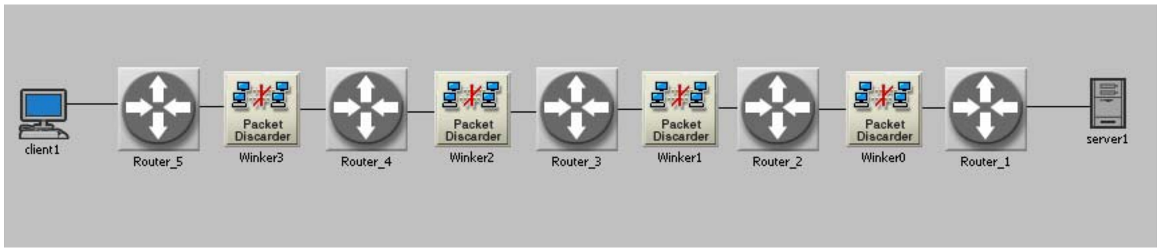


Figure 4.11: Multiple Challenged Link Topology

Figure 4.12 displays the number of TCP aborts recorded for the scenario when the challenged links are not supported with **link proxy** routers. Aborts result from both failures of the communicating endpoints to establish a connection and initiate an FTP transfer as well as unconditional aborts by the TCP sender during file transfer. No distinction is made between the two cases as both are considered communication failures.

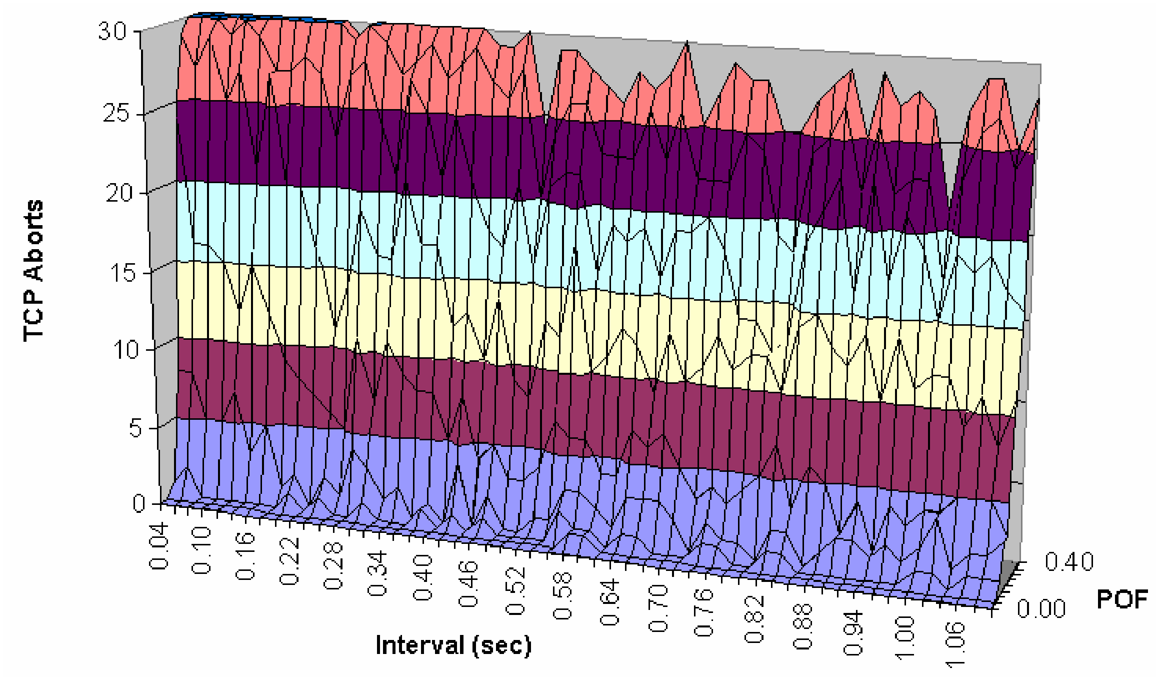


Figure 4.12: TCP Aborts Observed for Four Challenged Link using Standard Router Configuration

Connection aborts occur for individual link failure probabilities as low as 5%. Every interval experiences between 1.3 and 2.8 percent of the total number of aborts. Intervals less than 500 msec experience the greatest number of aborts and in general, the shorter the failure interval, the greater the probability of an abort. Unlike the single challenged link scenario, seed value does not appear to influence abort results.

Impressively, using **link proxy** routers on the challenged links enabled ETE TCP communication to succeed for every probability of failure and link failure interval simulated, even for the 40% probability of failure case where 94% of all standard router communications failed. ETE connectivity in the simulation is reduced to 13% for 40% individual link failure probability, so lack of any TCP communication aborts is very encouraging.

The average transfer time for non-aborted FTP transfers, using standard routers, is displayed in Figure 4.13 and the variance associated with the data is displayed in Figure 4.14. Under perfect conditions, transmission of the 20MB file requires 1.737 seconds. As was the case for a single challenged link, the primary trend is the shorter the link failure interval, the greater the impact to ETE TCP communication. Increasing the probability of link failure also increases the communication time. Note that the results are somewhat skewed by the large number of unsuccessful FTP transfers. The presented data represents the "lucky" TCP connections.

The increase in communication time from the baseline 1.737 seconds is attributed to same three factors previously discussed, however the effect is much more pronounced over multiple challenged links. Back to back TCP sender timeout events and the exponential increase of backoff time between successive retransmission timeout events are severely hampering ETE throughput performance.

The same valleys noted in the transfer time results of Figure 4.5 are again present at 500 and 1000 msec. Additional valleys are present in the data at 640 msec and between 820 and 960 msec; however they are artifacts of the data arising from the discount of non-successful TCP connections. The variance of the transfer time is quite high for almost every probability of failure and interval pairing, suggesting that more data samples should be collected for a higher confidence in the mean values.

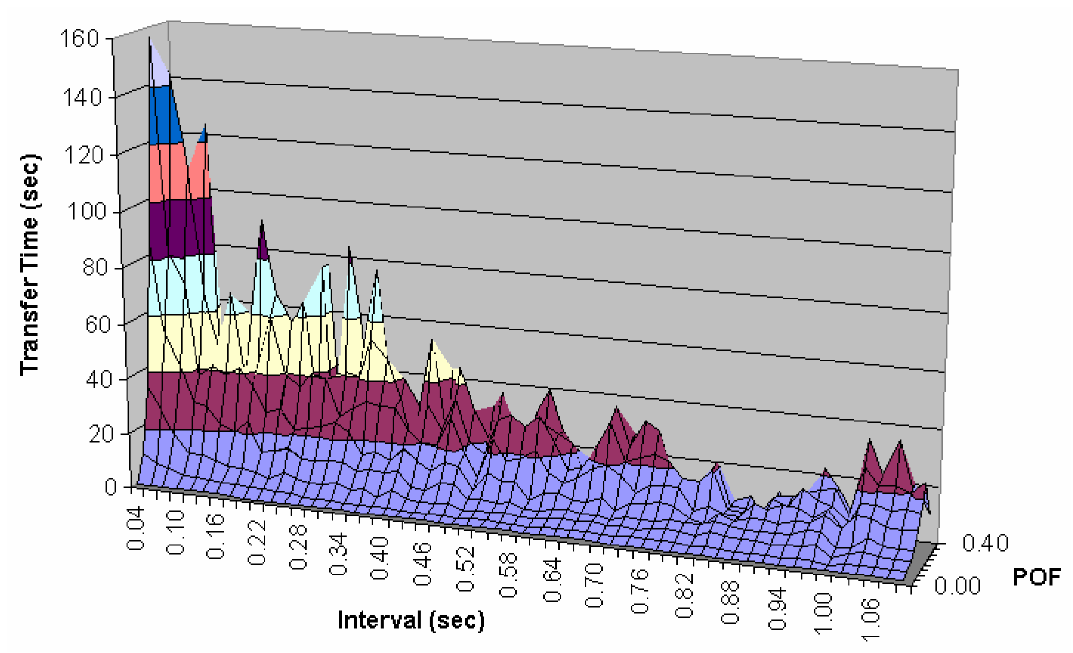


Figure 4.13: Average FTP Transfer Time of 20MB File over Four Challenged Links using Standard Routers

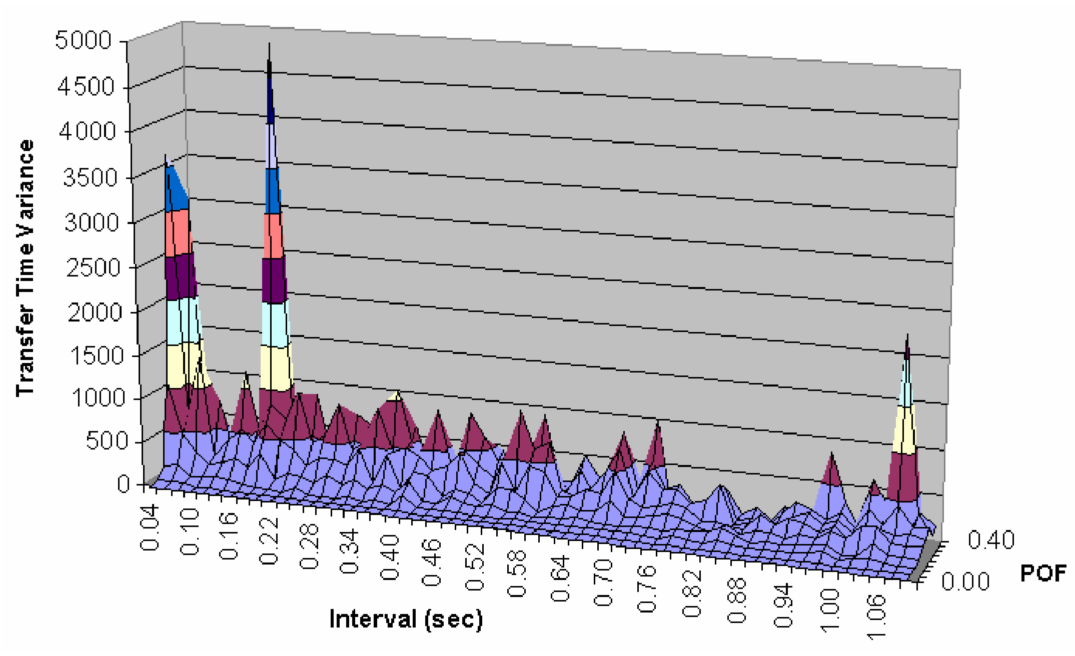


Figure 4.14: Variance of Average FTP Transfer Time of 20MB File over Four Challenged Links using Standard Routers

The mean transfer time for FTP transfers using **link proxy** routers is displayed in Figure 4.15 and the variance associated with the data is displayed in Figure 4.16. Under perfect conditions, transmission of the 20MB file requires 1.737 seconds. The longest mean transfer time noted was 26.6 seconds, recorded for 40% probability of failure and 10 msec failure interval.

At individual link failure probabilities of 25% and greater, corresponding to ETE availability of 32% and less, peaks in ETE mean transfer time are noted at approximately 10 msec intervals. The peaks become more pronounced as individual link failure probabilities increase. Close inspection of the simulation data reveals several test points with ETE transfer times greater than 2 standard deviations from the mean.

The variance of the transfer time data is well behaved for link failure probabilities less than 35%, with occasional minor spikes at the aforementioned 10 msec intervals. For 35% and 40% link failure probability of failure however, mean transfer time variance spikes are directly correlating with the noted increase in ETE transfer time. Future effort should be expended to determine the source of ETE transfer time increase.

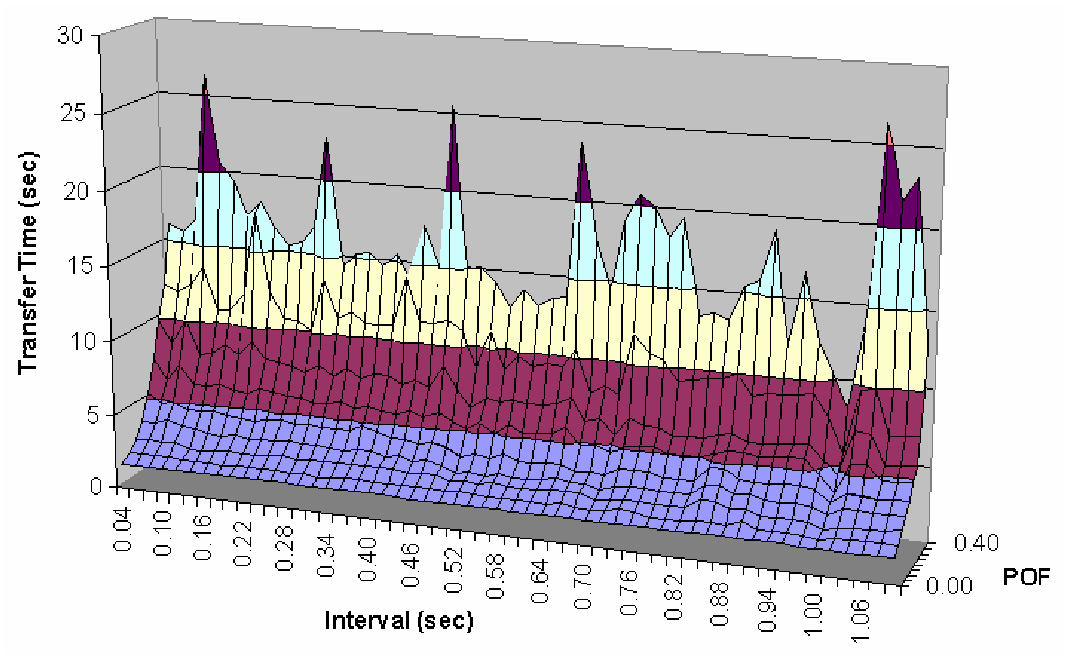


Figure 4.15: Average FTP Transfer Time of 20MB File over Four Challenged Links using Link Proxy Routers

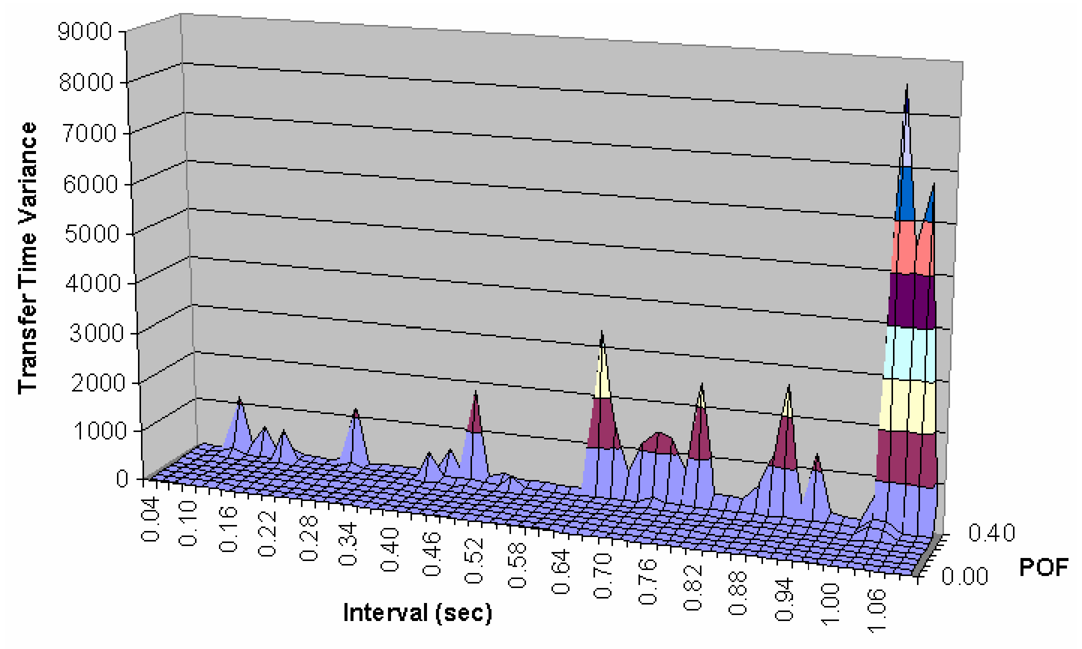


Figure 4.16: Variance of Average FTP Transfer Time of 20MB File over Four Challenged Links using Link Proxy Routers

The overall improvement observed for each probability of failure and link failure interval pair is displayed in Figure 4.17. Note the order of magnitude improvement in several cases. Again, the greatest improvement is noted for link failure intervals less than 500 msec and increased individual link probability of failure rates. For comparison purposes, we must remove simulation events, which resulted in TCP failure using standard routers in the case of **link proxy** routers. This removal, while critical for side by side comparison, reduces the presented impact, especially for higher probability of failure.

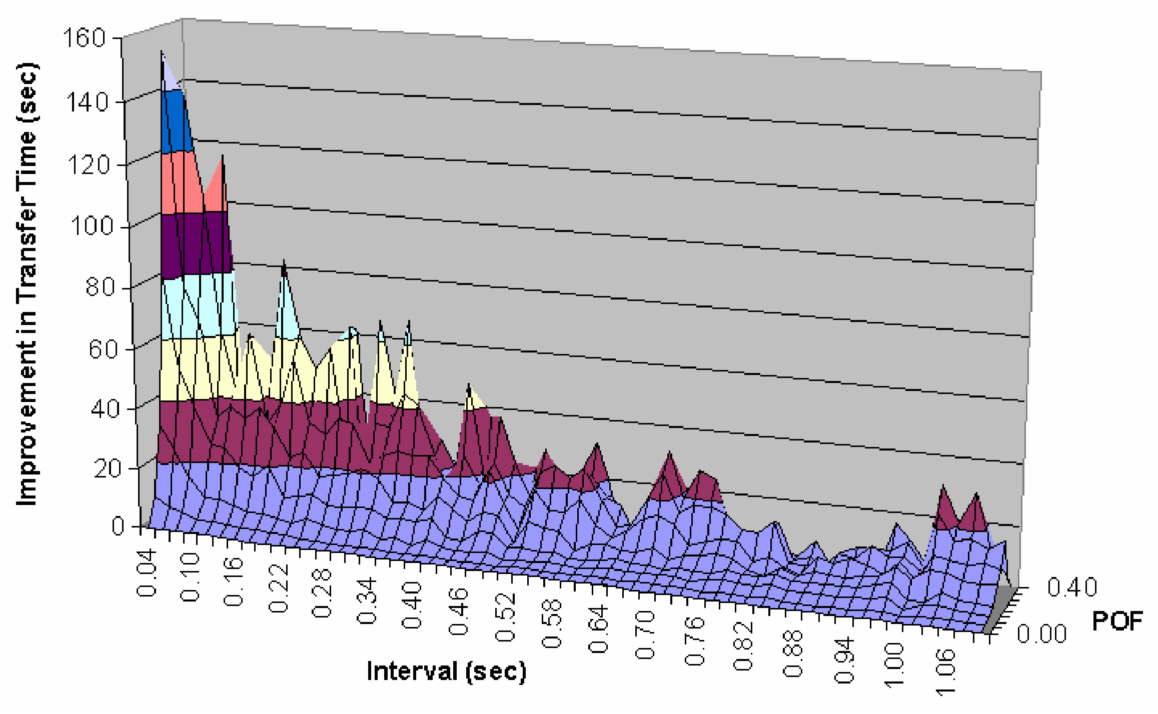


Figure 4.17: Improvement in Mean FTP Transfer Time of 20MB File over Four Challenged Links through Use of Link Proxy Routers

The difference in utilized ETE bandwidth, adjusted for the expected ETE bottleneck introduced by the four challenged links is displayed in Figure 4.18. As expected, using **link proxy** routers again enables increased bandwidth usage across the board due to near-immediate loss event detection. Dips to zero or near-zero improvement are introduced by exclusion of TCP communication failures using standard routers. Excluding failures from the calculation leaves very few standard router file transfers available for ETE throughput calculation, most of which experienced little or no communication disruption. Likewise, the throughput greater than one artifact noted at 35% probability of failure and 1060 msec failure interval is introduced by the ability to use only three standard router data points, two of which exhibit poor performance.

Again, the most notable gains occurred in intervals less than 500 msec with a downward trend as the interval nears 500 msec. The downward trend however is not as notable as in the single challenged link case because the combination of failed link probing and forward custodial buffering principle ensures that available bandwidth is used if data is available for link transit.

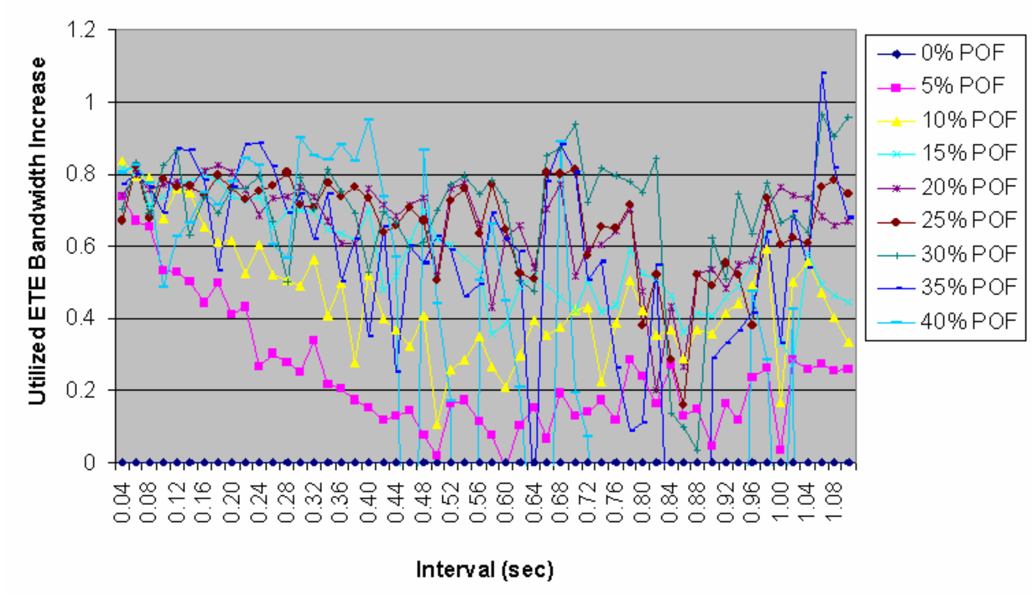


Figure 4.18: Improvement in Utilized ETE Bandwidth over Four Challenged Links Using Link Proxy Routers

4.3.4 Questions 4 and 5

Does implementing **link proxy** routers negatively affect non-modified routers within the network? Does supporting TCP byte streams with **link proxy** routers negatively affect fairness of other TCP byte streams within the network? To appropriately answer these questions, the topology shown in Figure 4.19 was simulated using a subset of the factors described in Table 4.3. The scenario consists of a three clients simultaneously initiating a TCP connection at simulation time 15 seconds and simultaneously requesting an FTP transfer of a 20MB file from three remote servers. TCP communication sessions exist between Client 1 and Server 1(flow 1), Client 2 and Server 2 (flow 2), and Client 3 and Server 3 (flow 3). A mixed router topology is used with standard routers (Routers 1 and 4) and **link proxy** routers (Routers 2 and 3) supporting a single challenged link. Simulations are performed using challenged link probability of failure ranging from 0 to 40 % at 10% intervals and 40 to 500 msec failure interval at 20 msec increments. Each probability of failure and failure interval pair is simulated with 30 repetitions via unique seeds.

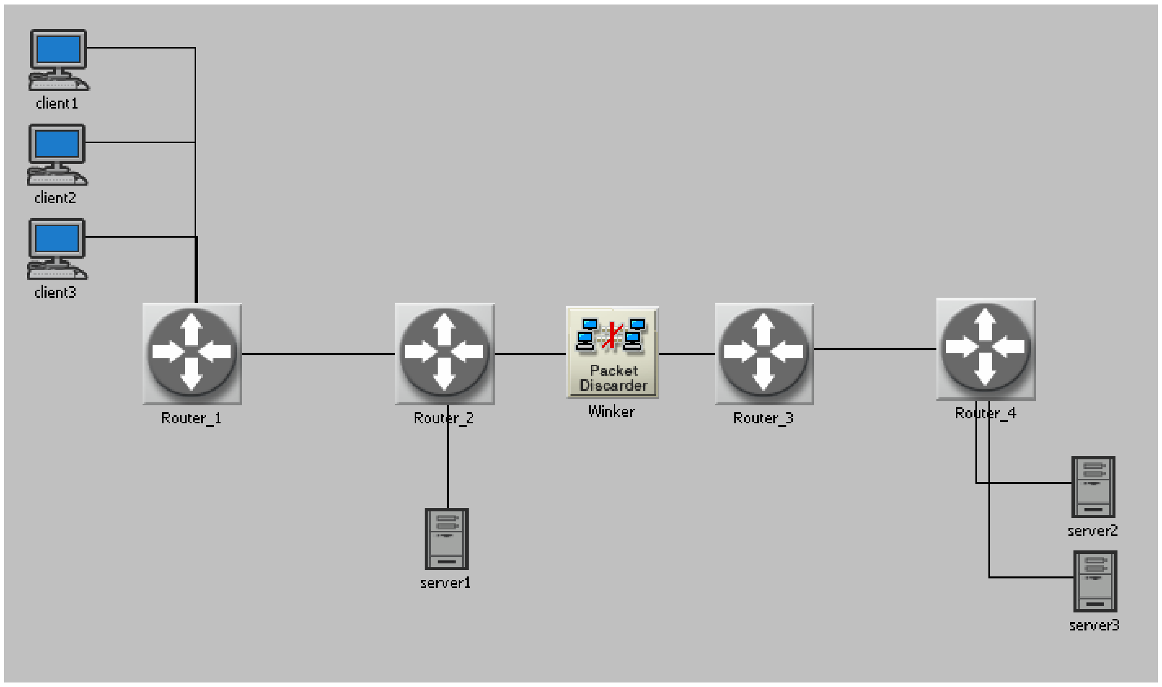


Figure 4.19: Fairness Evaluation Topology

The simulation topology tests the ability of standard and **link proxy** routers to work together supporting TCP flows in a static routing environment. Other routing schemes are outside the scope of this effort, however **link proxy** routers make no modification to routing algorithm processing, allowing the simplification. The topology also allows an evaluation of fairness between the three TCP flows transiting the network. In this scenario, fairness is defined as flow 1 bandwidth utilization between routers 2 and 3 suffering no degradation as a result of the presence of **link proxy** routers supporting

TCP flows 2 and 3. Additionally, TCP flows 2 and 3, transiting the challenged link, should exhibit near-identical bandwidth utilization between the same routers and total communication time performance across varying degrees of challenged link degradation.

The baseline performance for all three flows in this scenario is determined by simulation with no failures across the challenged link. Flow 1 requires 5.101 seconds for the 20MB file transfer and flows 2 and 3 each require 5.174 seconds. Bandwidth utilization between routers 2 and 3 is 32.9%, 32.4%, and 32.4% for flows 1, 2, and 3 respectively.

At increasing challenged link probabilities of failure, the challenged link effective bandwidth is reduced, placing less demand on downstream routers. As displayed in Figure 4.20, once the challenged link suffers reduced availability, flow 1 readily uses more available bandwidth on the link between routers 2 and 3. Accordingly, the bandwidth used by flow 2 is represented by Figure 4.21. Note that the y-axis is plotted in reverse order for easy viewing. Bandwidth utilization by flow 3 is indistinguishable from that of flow 2.

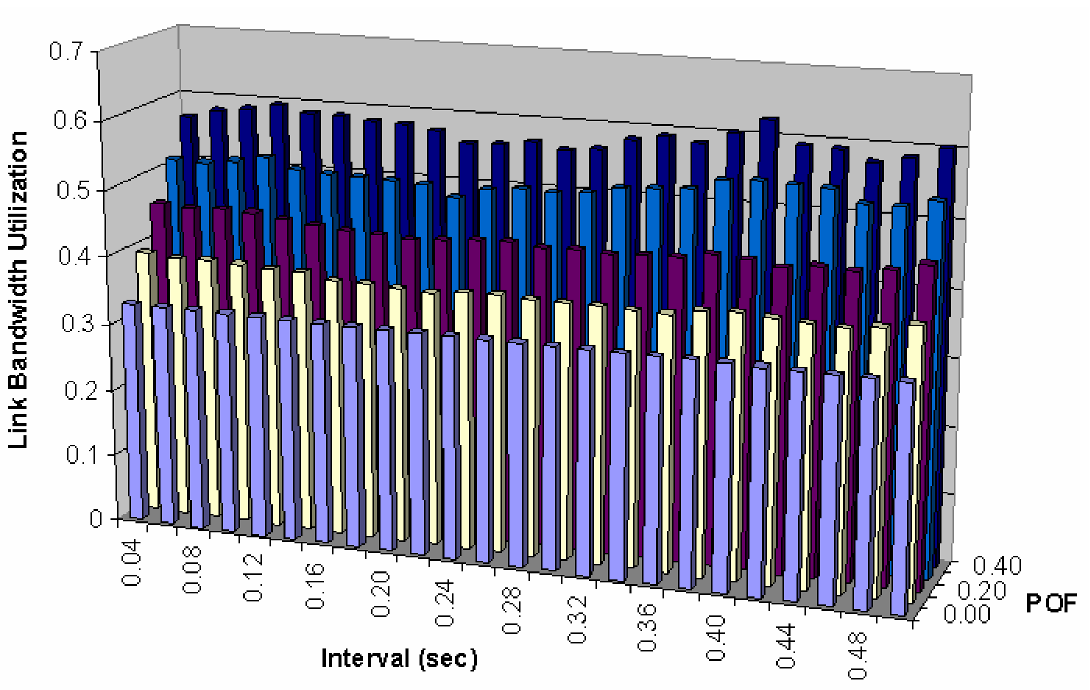


Figure 4.20: Router 2 – Router 3 Link Bandwidth Utilization of Flow 1

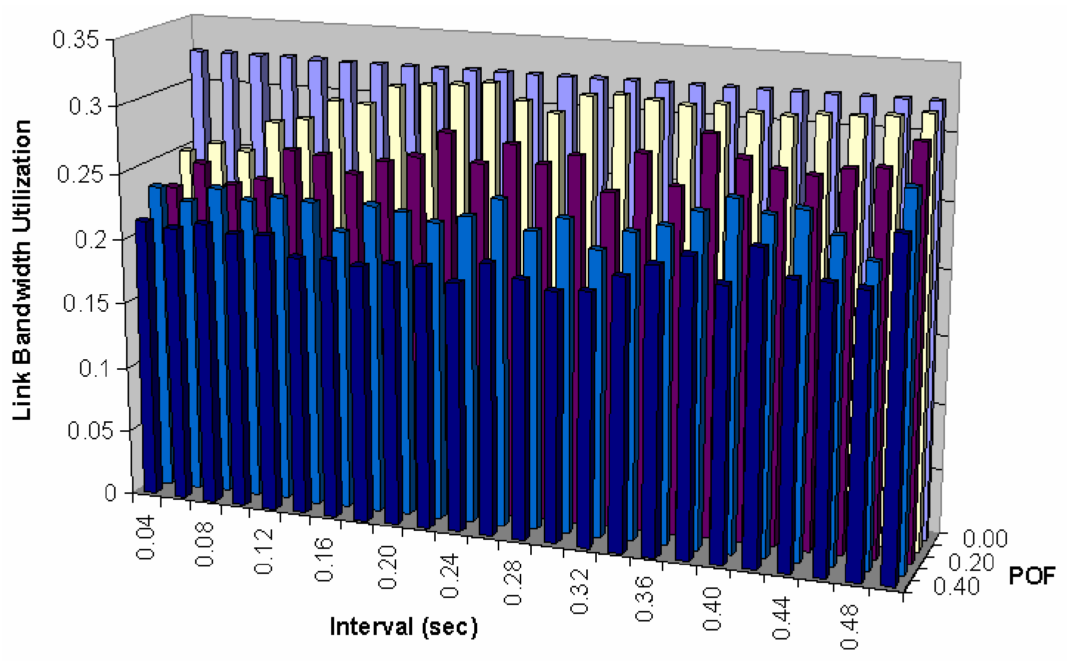


Figure 4.21: Router 2 – Router 3 Link Bandwidth Utilization of Flow 2

For all simulation test points, as the probability of failure of the challenged link increases, flow 1 ETE file transfer time diminishes due to additional bandwidth becoming available for use on the link between routers 2 and 3. Accordingly, flows 2 and 3 require additional transfer time from restricted shared bandwidth on the challenged link. This is indeed the observed trend and as Figure 4.22 shows, transfer time performance of flows 2 and 3 is virtually identical, showing no preferential treatment among flows.

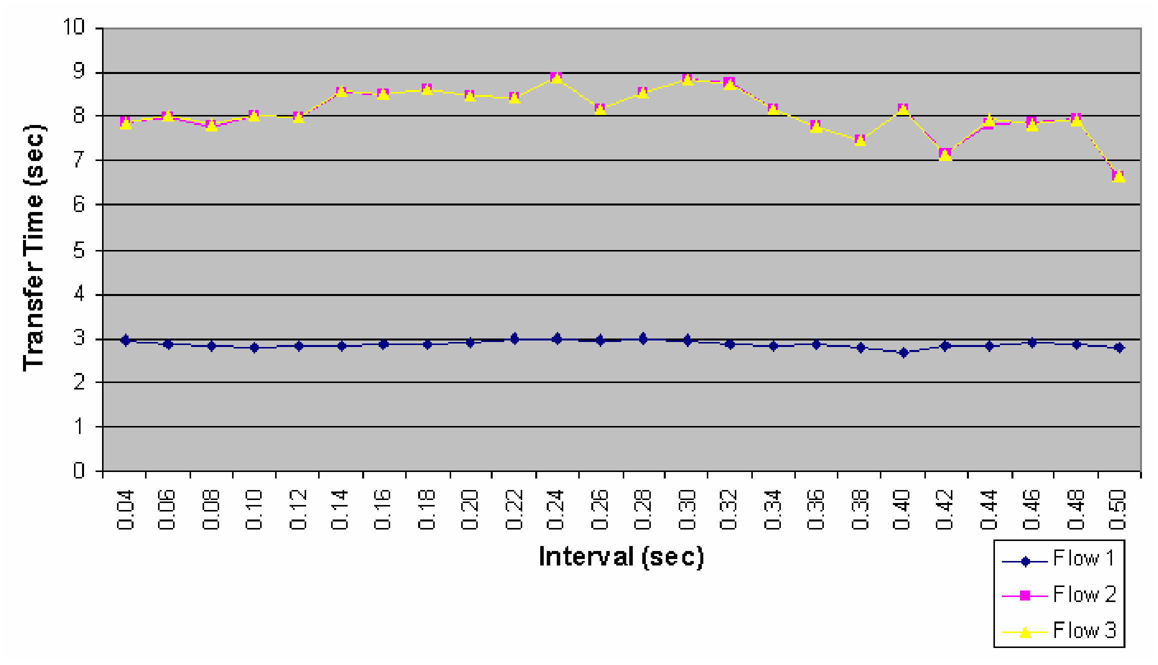


Figure 4.22: Mean FTP Transfer Time of 20MB File over Single Challenged Link (40% POF) with Mixed Router Topology

4.4 Summary

This chapter presents the results of simulations with challenged links using both standard and **link proxy** routers.   A demonstration is made of the developed model's ability to decouple channel loss from congestion loss, without modification to a TCP sender. Additionally, the ability to mix standard and **link proxy** routers without negatively impacting non-buffered flows within the network is demonstrated. Comparisons of ETE network channel bandwidth utilization were presented for single and multiple challenged link scenarios, showing that the developed model discovers and utilizes available bandwidth on problematic links. The next chapter discusses the relevant conclusions of this research and suggestions for future research.

V. Conclusions and Recommendations

5.1 Chapter Overview

This chapter presents the final conclusions of this thesis and discovers implications for future challenged network environments. Several suggestions for future research, including model modifications, are also discussed.

5.2 Conclusions of Research

It should be anticipated that potential adversaries will attempt to deny our country the use of information superiority assets. Enemy jamming of communications frequencies will impact existing wireless communication assets on the battlefield. Packet switched communication networks in such a domain can utilize an intermediate buffering strategy as outlined in this study to overcome short (and, with TCP modifications, long) disruption periods within the challenged environment.

This research has shown that successful TCP communication is severely hampered when multiple challenged links exist between two communicating endpoints. Introducing even a low probability of failure on each link manifests degradation of ETE TCP connectivity and connection maintenance issues. TCP's ability to even establish a connection is highly questionable in such environments without network assistance. Link-layer solutions to solve the challenged environment problem may not be sufficient to overcome the situation where multiple challenged links exist, especially for long duration outages in the millisecond region. Accordingly, adding complexity to network routing infrastructure in the form of intermediate buffering will be required for future reliable TCP communication in challenged environments. It appears that even a modest investment of memory cost can provide a significant improvement in performance, provided that reasonable protocols can be sufficiently tuned to the environment. A transport layer TCP aware helper protocol using methods such as the developed model employs can successfully overcome end to end connectivity as low as 10%.

Implementing **link proxy** routers can hide some, but not all, non-congestion losses from a TCP sender without modifications to the TCP endpoints. Short individual link failure outages are easily discovered and handled locally by **link proxy** routers, resulting in masking non-congestion losses from the TCP sender. The ability to conceal longer link failure channel losses however is highly dependent on the TCP sender settings for minimum allowed retransmission timer value. Any combination of individual link failures, especially a rolling failure from destination to source, exceeding the TCP sender's calculated retransmission value will result in invocation of the *slow start congestion control* algorithm and an immediate throughput loss.

While using link proxies will not always prevent an unmodified TCP sender from invoking *congestion control*, ETE TCP communication throughput is improved in both single and multiple challenged link environments for investigated failure intervals. Over a single challenged link, mean communication time using standard routers required up to thirteen times that required using **link proxy** routers. When four challenged links are simulated, a performance gain of more than five times is noted. The performance gain over four challenged links is an under-estimation due to data exclusion of TCP communication failures using standard routers. A **link proxy** router's direct connection to the challenged link allows near real-time loss discovery for TCP flows transiting the link. Upon loss detection, a **link proxy** can politely resend all locally cached segments lost on the challenged link. Performing a resend action locally avoids requiring the TCP sender to idly wait for a retransmission timeout when no acknowledgments are going to be received. Additionally, a segment has already incurred propagation and queuing delays to arrive at the point of loss. Handling loss retransmission locally avoids re-incurring such costs.

Integrating **link proxy** routers into the network does not appear to negatively affect standard routers within the network. The proposed protocol does not change routing protocol semantics and is hidden from network layer processing. Additionally, TCP flows supported by **link proxy** routers enjoy no special advantage or preference by the network once the challenged link has been successfully navigated. On the contrary, supported flows receive their fair share of non-challenged link resources because segments are present for routing and eventual delivery when they would be lost otherwise. Failure to handle loss events locally places additional burden on the network as a whole, requiring resource utilization re-transporting lost segments to the challenged link, only to be potentially lost again.

Utilizing **link proxy** routers adds overhead to the network in the form of intermediate acknowledgements, persist request messages, and persist response messages. These messages are extremely small in nature and the impact to the overall health of a network is negligible. The overhead is worth paying however in that it supports communication where it would otherwise fail, negating the purpose of portions of the network altogether.

5.3 Recommendations for Future Research

While performing this study, several topics of concern surfaced that should be given consideration in future research. Most of these topics are related to expansion of the developed model for increased performance, however, security considerations, flow control, and scalability should all be considered.

5.3.1 Proxy Router Model Modifications

The following features should be considered for **link proxy** router model and protocol implementation.

5.3.1.1 Inter-Router Congestion Control

As discussed in Chapter Two, TCP assumes that loss events occur only at intermediate routers within the network.   TCP invokes *congestion control* in response to such loss events and reduces the number of outstanding segments within the network. Using the proposed **link proxy** routers prevents a TCP sender from invoking *congestion control* by masking invocation mechanisms, however genuine network congestion within network routers is also masked.

Segments dropped at a router due to congestion will not be acknowledged via intermediate or receiver acknowledgments. Lack of an acknowledgment is interpreted by the sending **link proxy** as a loss event causing an immediate resend of the lost segments. Such behavior is a partial disregard for the requirements of TCP RFC standards which endeavor to prevent network over-utilization, i.e., reduce congestion within the network.

In effect, utilizing the existing **link proxy** router impairs a TCP sender's ability to misinterpret channel losses as congestion, but is also masking congestion from the TCP sender when it truly exists.

This discrepancy can possibly be alleviated by including an additional field in intermediate acknowledgments and persist response messages that advertises current routing buffer congestion notification status. Such information can be used to discriminate congestion loss from channel loss, invoking a small backoff before resending lost segments.

Alternately, congestion could be explicitly estimated or measured, perhaps following the work of Stuckey [16], and used to invoke *congestion control* directly.

5.3.1.2 Window Scaling Support

Current classical TCP implementations allow the amount of unacknowledged data within the network to be the minimum of the sender derived *congestion window* parameter or advertised *offered window* value. Unfortunately, the 16 bit *offered window* field in a TCP header packet limits the *offered window* to 65535 bytes. Simulations performed for this study show that the 655535 byte threshold is quickly reached by the TCP sender, artificially limiting TCP throughput when network capacity exists.

Consideration should be given to enabling support for utilizing the RFC 1323 [17] window scale option, which would enable more data to be in transit between communication endpoints. Such support also requires both TCP sender and receiver capability support, as window scaling requires additional state data maintenance and exchange by the communicating endpoints.

5.3.1.3 Buffer Management

The developed model uses a simplified memory management approach that allocates available memory to TCP flows a first come first served basis. This study ensured ample memory was made available such that memory resource contention was never an issue. The developed model also provides no support for unconditionally aborted TCP flow discovery and memory reclamation, though such modifications can be easily implemented. It is unreasonable however to expect **link proxy** routers to provide infinite buffer capacity. Accordingly, the centralized memory manager built into the model requires a proper memory management routine be developed for efficient allocation of limited memory resources.

Proper memory management however requires some form of flow priority knowledge be made available. It is unclear at this time what means of conveying such priority to a **link proxy** router should be used, but consideration should be given to establishing a new TCP option to convey such information to the routers. Establishing a new option would require communication endpoint modifications.

5.3.1.4 Fault Tolerance

Link proxy routers provide a loose form of TCP segment custody acceptance as segments pass through a network. Receipt of a destination or intermediate acknowledgement for the data carried in a segment is sufficient to know that the destination, or a **link proxy** router closer to the destination, has received the segment in its entirety. Receipt of acknowledgements updates flow state data that is used for forwarding or discarding duplicate resent segments from a TCP sender.

It is theoretically possible that downstream buffer management algorithms however may dictate that intermediately acknowledged segments be purged from buffers. If such an event occurs at a **link proxy** experiencing loss events, it is entirely possible that the only cached segment within the network is lost, requiring an eventual resend from the TCP sender. Presently, the developed model provides no capability to re-buffer segments that have been intermediately acknowledged. Such segments are simply forwarded and subjected to potential loss with no localized **link proxy** recovery mechanism.

Link proxy router fault tolerance should be implemented to deal with intentionally dropped segments of this type. Should a **link proxy** unconditionally purge unacknowledged segments as instructed by the memory manager, it should generate a negative acknowledgment message to upstream link proxies indicating such. Receipt of such a message can be used to "roll-back" the intermediate acknowledgment state data for proper segment buffering. Should this message be lost as a result of channel loss, subsequently received intermediate acknowledgments would show that a gap in received data exists, necessitating data resends.

5.3.2 Challenged Link Limits

The ability of **link proxy** routers to overcome multiple challenged links was investigated with probabilities of failure down to 40%. Consideration should be made to find a breaking point at which **link proxy** routers no longer perform adequately. It is theoretically possible for **link proxy** routers to support a TCP connection that never has a functioning ETE connection without short duration outages. For example, four links failing out of phase in a rolling pattern from source to destination would never have ETE connectivity, but **link proxy** routers could handle such a situation with ease. Investigating a breaking point may highlight unknown weaknesses in the developed model that should be pursued for additional ETE TCP connection reliability.

5.3.3 Security Considerations

Information security has not been addressed in this thesis. A fundamental assumption made in this study is that IP and TCP header information is readily available for inspection within intermediate routers. It should be stressed that inspection of TCP payload data within segments is not required. Many forms of data encryption exist and an exhaustive survey of standards was not performed. Future consideration should be given to transport layer encryption standards such as IPsec, outlined in RFC 4301 [18], which encrypts IP datagrams between communication endpoints, making TCP header information unavailable.

Another concern is physical custody of routers with intermediate buffering capability. For the purposes of this study, it is envisioned that **link proxy** routers will be utilized within strategically placed backbone routers under military jurisdiction.

5.3.4 Custodial TCP Flow Control

Link proxy buffer capacity enables the concept of routers accepting custody, or responsibility for delivery, of a received TCP segment. **Link proxy** routers observe new TCP flows and allocate buffer space to support them. If challenged links are experiencing little or no channel losses, buffer space usage is extremely minimal. Moderate and heavy channel losses however would require considerable buffer capacity, especially if multiple high capacity TCP flows, possibly using window scaling, are utilizing the challenged link.

A **link proxy** router, being aware of local link performance and individual TCP flow buffer capacity requirements, could advertise to upstream routers that buffer capacity is decreasing. Such an advertisement could be made by modifying the TCP header window size field in acknowledgments to match remaining available buffer space. Upon receipt of these acknowledgments, **link proxy** routers could inspect and utilize the information for the purpose of deciding when to "back off" on segment transmissions. In this fashion, each **link proxy** enroute to the destination can buffer a significant amount of data within the network, ready to use available bandwidth as soon as it becomes available.

An additional side benefit of implementing this strategy is that advertising a window size of zero places a TCP sender into a persist state, avoiding *congestion control* invocation. Additional network overhead is required however to recover from advertising zero window size since probes will need to be made to discover that capacity is available.

5.3.5 Scalability

This thesis focused on using **link proxy** routers in a non-congestion environment. It is highly suggested that future research explicitly address scenarios where network congestion exists. Link proxies currently perform timeout event handling on a per-flow basis; however it is anticipated that some mechanism will be required to support timeout event handling for a large numbers of TCP flows. Upon a short duration link failure, all flows transiting the link will experience a timeout event resulting in a mass resend of all cached segments in the **link proxy**. Repeated timeout events will resend even more cached segments as additional segments arrive from upstream. Fairness among TCP flows could be an issue in such an environment and should be investigated further.

5.3.6 Environmental Assumption Relaxation

Several assumptions made during implementation of this thesis can be relaxed in future research. An expansion of model support for additional transport layer TCP functionality such as interactive communication as well and UDP buffering support should be given consideration.   Integrating routing protocols other than static should be investigated with heavy consideration given to mobile routing environment supportive algorithms. Such algorithms may gain momentum when appropriately linked with this transport layer helper protocol.

The protocol implemented in this thesis is claimed to be shared medium capable, but no investigation has been made to support such. A **link proxy** is located between the network layer routing protocol and link layer transmission protocol, placing no restrictions on either. In theory, accommodations are made to enforce a form of politeness that does not overburden the link layer and dominate the medium in event of a loss.

Finally, this thesis was performed with no processing delay component associated with actions required by a **link proxy** in supporting a flow. Some form of processing delay should be modeled which places realistic limits on the number of transactions that can be performed by a **link proxy** within a finite period of time. Memory functions such as reading and writing tend to dominate computing transactions and the delay associated with buffering activities could be abysmally slow. The distributed **link proxy** router architecture should help somewhat in this regard, but the true performance gain from link such routers can only be measured with a processing delay component.

5.4 Summary

This thesis demonstrates that future network infrastructure should provide some form of intermediate buffering capability at nodes adjacent to challenged links. TCP's capability to establish and maintain a connection in the presence of a single or multiple challenged links can be severely degraded, especially when short duration link failures on the order of 200 msec or less are highly probabilistic. A transport layer TCP helper protocol with intermediate buffering capability, implemented in network routers, can significantly improve TCP's reliability and performance in such environments.

The forward deployed warfighter requires a reliable network infrastructure capable of providing timely and accurate information, from any source. The GIG is required to provide such reliability and this research shows that appropriately applied intermediate buffering is a reliability and performance enabler. The DoD should consider placing intermediate buffer capable routers, such as those developed for this thesis, within strategically forward deployed assets in its transition to a Net-Centric force.

VI. Bibliography

1. Department of Defense, *Department of Defense Directive 8100.1 - Global Information Grid Overarching Policy*. September 19, 2002.
2. John G. Grimes, DoD Chief Information Officer, *2006 Department of Defense Chief Information Officer Strategic Plan - Version 1*, 2006.
3. Reynolds, M.B., *Mitigating TCP Degredation Over Intermittent Link Failures Using Intermediate Buffers*, in *Department of Electrical and Computer Engineering*. 2006, Air Force Institute of Technology: Wright-Patterson AFB.
4. Hari, B., et al., *A comparison of mechanisms for improving TCP performance over wireless links.* IEEE/ACM Trans. Netw., 1997. **5**(6): p. 756-769.
5. Sushant, J., F. Kevin, and P. Rabin, *Routing in a delay tolerant network*, in *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*. 2004, ACM Press: Portland, Oregon, USA.
6. Postel, J., *IETF RFC 793: Transmission Control Protocol*. September 1981.
7. Stevens, W.R., *TCP/IP Illustrated Volume 1*. August 2004 - 25th Printing: Addison Wesley.
8. M. Mathis, J.M., S. Floyd, A. Romanow, *IETF RFC 2018: TCP Selective Acknowledgement Options*. October 1996.
9. Hari, B., et al., *Improving TCI/IP performance over wireless networks*, in *Proceedings of the 1st annual international conference on Mobile computing and networking*. 1995, ACM Press: Berkeley, California, United States.
10. Chi Ho Ng, J.C., Ljiljana Trajkovic. *Performance Evaluation of TCP over WLAN 802.11 with Snoop Performance Enhancing Proxy*. in *OPNETWORK*. 2002. Washington, D.C.
11. Swastik Kopparty, S.V.K., Michalis Faloutsos, and Satish K. Tripathi. *Split TCP for Mobile Ad Hoc Networks*. in *Proceedings of IEEE GLOBECOM*. 2002. Taipei.
12. Guang Yang, R.W., Mario Gerla, M. Y. Sanadidi. *TCP Bulk Repeat for Heavy Random Losses: A Performance Analysis*. in *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'04)*. 2004. San Jose, CA.
13. www.opnet.com.
14. V. Paxson, M.A., *IETF RFC 2988: Computing TCP's Retransmission Timer*. November 2000.
15. M. Allman, V.P., W. Stevens, *IETF RFC 2581: TCP Congestion Control*. April 1999.

16. Stuckey, N., *Stochastic Estimation in Control of Queues Within a Computer Network*, in *Department of Electrical and Computer Engineering*. 2007, Air Force Institute of Technology: Wright-Patterson AFB.

17. V. Jacobson, R.B., D. Borman, *IETF RFC 1323: TCP Extensions for High Performance.* May 1992.

18. S. Kent, K.S., *IETF RFC 4301: Security Architecture for the Internet Protocol.* December 2005.

Vita

Major Duane F. Harmon graduated from Niceville Senior High School in Niceville, Florida. He entered undergraduate studies at the University of South Alabama, where he graduated with a Bachelor of Science in Electrical Engineering in March 1996.

He was commissioned through the Detachment 432-AFROTC at the University of South Alabama and assigned to the ICBM System Program Office at Hill AFB where he served as the Lead Engineer for ICBM Codes and Targeting Software. In February 1999, he was assigned to the 84th Test and Evaluation Squadron at Tyndall AFB, Florida as the Lead Missile Engineer responsible for evaluating weapon integration and post-launch support for the F-15 and F-16. In April 2001, he was handpicked to be the F-15 Flight Commander, responsible for the unit's Operational Test and Evaluation of F-15 Suite 4 and Suite 5 Operational Flight Programs and integrated weapon system evaluation. In July 2002, he reported to the 413th Flight test Squadron at Edwards AFB where he served as a Project Officer responsible for electronic warfare testing for the United States and allied services. In May 2005, he entered the Graduate School of Engineering and Management, Air Force Institute of Technology. Upon graduation, he will be assigned to the Air Force Research Laboratory at Wright Patterson AFB, OH.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)* 22-03-2007	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From – To)* May 2005 – March 2007

4. TITLE AND SUBTITLE Overcoming TCP Degradation in the Presence of Multiple Intermittent Link Failures Utilizing Intermediate Buffering	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Harmon, Duane F., Major, USAF	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GE/ENG/07-11
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Dr. David R. Luginbuhl (703) 696-6207 AFOSR 875 North Randolph Street Arlington, VA 22203-1768 david.luginbuhl@afosr.af.mil	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
It is well documented that assumptions made in the popular Transmission Control Protocol's (TCP) development, while essential in the highly reliable wired environment, are incompatible with today's wireless network realities in what we refer to as a challenged environment. Challenged environments severely degrade the capability of TCP to establish and maintain a communication connection with reasonable throughput.

This thesis proposes and implements an intermediate buffering scheme, implemented at the transport layer, which serves as a TCP helper protocol for use in network routing equipment to overcome short and bursty, but regular, link failures. Moreover, the implementation requires no modifications to existing TCP implementations at communicating nodes and integrates well with existing routing equipment. In a simulated six-hop network with five modified routers supporting four challenged links, each with only 60% availability, TCP connections are reliably established and maintained, despite the poor link availability, whereas 94% fail using standard routing equipment, i.e., without the TCP helper protocol.

15. SUBJECT TERMS
TCP, buffering, network layer, transport layer, congestion control, OPNET, channel loss

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Scott Graham, Major, USAF
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	120	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-6565, ext 4918 Scott.Graham@afit.edu